Measurable Targets for Scalable Reasoning
Atanas Kiryakov, Ontotext Lab, Sirma Group Corp.
27th of November 2007

Abstract: Defining adequate performance targets is crucial for the development of scalable semantic repositories that combine VLDB management with reasoning. I start with an analysis of the relevant tasks (especially, loading and query answering), performance criteria and problem dimensions (e.g. speed, complexity, size). The major contributions of this document are the analysis of the state of the art in scalable reasoning and the definition of measurable performance targets to be met by year 2010. This analysis is based on published results of several of the most scalable engines: ORACLE, AllegroGraph, DAML DB, Openlink Virtuoso, and BigOWLIM. The targets are defined with respect to two of the currently most popular performance measuring sticks: the LUBM repository benchmark (and its UOBM modification) and the OWL version of UNIPROT – the richest database of protein-related information.

Contents
1 Prologue.......................................................................................................................... 2
2 Introduction...................................................................................................................... 3
  2.1 Reasoning Strategies................................................................................................. 3
  2.2 OWL Layering and Variations.................................................................................. 4
3 Semantic Repository Tasks and Performance Factors................................................. 8
  3.1 Performance Dimensions....................................................................................... 9
  3.2 Nowadays Measuring Sticks: LUBM and UNIPROT............................................... 9
4 State of the Art of Scalable Reasoning.......................................................................... 11
  4.1 Scalability of DL Reasoning ................................................................................. 11
    4.1.1 UOBM Benchmark Results............................................................................. 11
    4.1.2 SHER and ABox Summarization.................................................................. 13
  4.2 Systems and Datasets .............................................................................................. 15
  4.3 Loading Performance.............................................................................................. 16
  4.4 Query Evaluation Performance.............................................................................. 18
5 Performance Targets...................................................................................................... 21
6 Conclusion....................................................................................................................... 22
7 References....................................................................................................................... 23

© Copyright Ontotext Lab, Sirma Group Corp. 2007
1 Prologue

This document presents initial work on defining measurable performance targets regarding scalable reasoning. The motivation came from the need for defining success criteria for project LARKC (Large Knowledge Collider). LARKC focuses on development of reasoning methods and infrastructure that can match the performance, scalability, and behavior of the web search engines: managing billions of documents/facts; real-time handling of large masses of users; incomplete and imperfect results; relevance ranking. An important issue is that the reasoning scheme and the infrastructure within LARKC have to allow for balancing and controlling two parameters: cost (computational resources allocated) and level of satisfaction.

The second incentive was Ontotext’s involvement in project RASCALLI, where we develop RDF engine with support for both logical reasoning and connectionist style spreading activation (on top of “weighted” RDF statements). We call this engine RDF Mind and use it to represent the mind of RASCALLI agents, “living” on Internet, in a way that resembles as much as practical the analogous cognitive functions of humans. In this context, there are two reasoning set-ups:

- “Working memory” - requires database-style (sound and complete) management of datasets of up to 100 million statements, and
- “Long-term memory” - requires incomplete reasoning against billions of statements, with requirements quite similar to those for web-scale reasoning in LARKC.

Last but not least, I wanted to define some criteria for the development of Ontotext’s semantic repository, named OWLIM. Competition is an important stimulus for development, but it works efficiently only in mature “markets”. Because of the early stage of development and the heterogeneity of the field, the leading semantic repository providers often appear without serious competition in specific “disciplines”, racing against clock over considerable periods. I believe that defining adequate performance targets is crucial for the development of scalable semantic repositories.

My goal is to provide a simplistic view on some aspects of the performance of semantic repositories, which allows for objective measurement. The reader should consider that the criteria defined here are shallow or ignorant with regards to substantial aspects of the reasoning tasks in focus. Most notably, no sufficient attention is paid to:

- Accuracy of the relevance ranking;
- Reasoning with non-tractable logical fragments.

Most of the state of the art analysis and benchmarking work included here is done in projects RASCALLI, TAO, and TripCom.
2 Introduction

Let us call “semantic repository” (SR) a tool, which combines the functionality of an RDF-based DBMS and an inference engine. A semantic repository can store data and evaluate queries, regarding the semantics of ontologies and metadata schemata.

Performance and scalability targets for SR are specified here based on analysis of the state-of-the-art engines. One shall consider however that direct comparison between a “classical” SR and a web-scale-and-spirit one is not feasible, because:

- Comparison between complete and incomplete reasoning and query evaluation has limited utility. Given a “tough” query to evaluate, an incomplete SR can consume as much time as it is allowed – more time delivers more and better quality results; in many cases there would be no upper limit of the time that can be used;
- The classical inference engines do not support relevance ranking of the results. Relevance ranking can be computationally expensive task, especially when context relevance and personalization are taken into account.

2.1 Reasoning Strategies

Many of the scalable semantic repositories nowadays perform reasoning that is based (more or less directly) on logical entailment rules. The two principle strategies for rule-based inference are, as follows:

- **Forward-chaining**: to start from the known facts (the explicit statements) and to perform inference in an inductive fashion. The goals of such reasoning can vary: to compute the inferred closure\(^1\); to answer a particular query; to infer a particular sort of knowledge (e.g. the class taxonomy).

- **Backward-chaining**: to start from a particular fact or a query and to verify it or get all possible results (bindings for the free variables), using deductive reasoning. In a nutshell, the reasoner decomposes (or transforms) the query (or the fact) into simpler (or alternative) facts, which are available in the knowledge base (KB) or can be proven through further recursive transformations.

Both of these strategies have well-known strong and weak points. Hybrid strategies are also possible and have proven to be efficient in many contexts.

Let us imagine a repository which performs total forward-chaining, i.e. after update to the KB the inferred closure is updated and kept available for query evaluation and retrieval. This strategy is known as *materialization*. In order to avoid ambiguity with various partial materialization approaches, let us call such inference strategy *total materialization*.

---

\(^1\) *Inferred closure* is defined here as follows: the extension of a KB (or a graph of RDF triples) with all the implicit facts (triples), which could be inferred from it, based on the enforced semantics.
The principle advantages and disadvantages of the total materialization are discussed at length in [2]; here we provide just a short summary of them:

- Upload/storage/addition of new facts is relatively slow, because the repository is extending the inferred closure after each transaction for modification. In fact, all the reasoning is performed during the upload;
- Deletion of facts is also slow, because the repository should remove from the inferred closure all the facts which are not true any longer;
- The maintenance of the inferred closure usually requires considerable additional space (RAM, disk, or both, depending on the implementation);
- Query and retrieval are fast, because no deduction, satisfiability checking, or other sorts of reasoning are required. Query evaluation becomes computationally comparable to the same task for relation database management systems (RDBMS).

Apart from the principle advantages and disadvantages of the reasoning strategies, their applicability is often pre-determined by the complexity of the semantics (ontology language) that has to be supported. Imagine a dataset, where large groups of individuals are interconnected though a transitive and symmetric property, e.g. fellow-citizen (as in [17]). If each citizen of a city with 100 thousand citizens is connected explicitly, say, with two others, there will be 200 thousand explicit fellow-citizen statements in the dataset. However, the inferred closure will contain about 10 billion statements! Obviously, total materialization can face troubles even with tractable logical formalisms.

### 2.2 OWL Layering and Variations

This subsection provides an overview of some variations of metadata and knowledge representation standards, relevant to the subject of scalable reasoning in web contexts. In my opinion, one of the major challenges to building scalable Semantic Web infrastructure is the expressivity of the underlying standards: RDF, RDFS, and OWL, defined in [16], [1], [4] respectively. The problem is that inference scalability and semantic compatibility were not primary concerns during the definition of some of the language layers (e.g. OWL Lite).

Even though RDFS can be considered a very simple Knowledge Representation (KR) language, it is already a challenging task to implement a repository for it, especially one that provides performance and scalability comparable to those of relational database management systems (RDBMS). Going up the “layers” of the Semantic Web specifications stack, the challenges for the repository engineers are getting more and more serious. Even the simplest official dialect of OWL (OWL Lite) is based on Description Logic (DL), whose complexity renders sound and complete reasoning against large knowledge bases (KB) theoretically impossible. Furthermore, the semantics of OWL Lite and OWL DL are incompatible with that of RDF(S), see [9]. This causes lack of “backward compatibility”. Imagine the situation when an application, using RDFS schemata and RDFS-compliant repository, should be “upgraded” to OWL. The evolution should start with replacement of RDFS schemata with OWL ontologies and adoption of a repository supporting (the
corresponding part of) OWL. Even the most direct translation (re-labeling the rdfs:Class
to owl:Class) of the schema, can lead to different inference and to inconsistencies.

Logical programming (LP) is a common name used for rule-based logical formalisms such as
PROLOG, Datalog, and F-Logic. OWL DLP is a non-standard dialect, offering a promising
compromise between expressive power, efficient reasoning, and RDFS compatibility. OWL
DLP is defined in [9] as the intersection of the expressivity of OWL DL and LP. In fact, OWL
DLP is defined as the most expressive sub-language of OWL DL, which can be mapped to
Datalog. OWL DLP is simpler than OWL Lite. The alignment of its semantics to the one of
RDFS is easier, in comparison with the OWL Lite and OWL DL dialects. Still, this mapping can
only be achieved through the enforcement of some additional modelling constraints and
transformations. A broad collection of resources related to OWL DLP can be found at
http://logic.aifb.uni-karlsruhe.de/.

In [23], ter Horst defines RDFS extensions towards rule support and describes a fragment of
OWL, more expressive than DLP. He introduces the notion of R-entailment of one (target)
RDF graph from another (source) RDF graph on the basis of a set of entailment rules R. R-
entailment is more general than the D-entailment used by Hayes, [12], in defining the
standard RDFS semantics. Each rule has a set of premises, which conjunctively define the
body of the rule. The premises are “extended” RDF statements, where variables can take
any of the three positions.

Horst extends and modifies the D-entailment rules from [12] in two steps as follows: D*
adds entailment support for literal data-types; pD* adds rules which provide partial support
for some OWL primitives, namely: FunctionalProperty, InverseFunctionalProperty,
SymmetricProperty, TransitiveProperty, sameAs, inverseOf, equivalentClass,
equivalentProperty, onProperty, hasValue, someValuesFrom, allValuesFrom,
differentFrom, disjointWith. The last two primitives are supported through
inconsistency rules which fire in case of the so-called P-clashes. It is important to
acknowledge that some of the primitives are only partially supported; the standard OWL
entailments related to someValuesFrom and allValuesFrom are supported only in one of
the directions (i.e. there is no full support for the iff-semantics of these OWL primitives).

I refer to this extension of RDFS as “OWL Horst”. As outlined in [23], it has several important
characteristics:

- It is a proper (backward-compatible) extension of RDFS. In contrast to OWL DLP, it
  puts no constraints on the RDFS semantics. The widely discussed meta-classes
  (classes as instances of other classes) are not disallowed in OWL Horst. It also does
  not enforce the “unique name” assumption;
- Unlike the DL-based rule languages, like SWRL, [14]and [18], R-entailment provides a
  formalism for rule extensions without DL-related constraints;
- Its complexity is lower than the one of SWRL and other approaches combining DL
  ontologies with rules, see section 5 of [23].
My observation is that most of the languages supported by the contemporary scalable semantic repositories are very similar to OWL-Horst, even when additional or alternative rules are used. Although additional rules make entailment more expensive, “safe” extensions raise the complexity by a constant. According to [23], a sufficient condition for remaining in the same class of complexity is that the rules should not introduce new blank nodes in the knowledge base. My practical hint is that for real-world OWL ontologies and datasets, it is sufficient to require that the above conditions shall only be enforced for the rules dealing with ABox reasoning (e.g. with instances). For instance, introducing a new blank node, which represents an auxiliary OWL restriction (or class), does not actually lead to exponential growth of the inferred facts.

![Complexity Diagram](image)

**Figure 1.** OWL Layering Map

Figure 1 presents a simplified map of the expressivity or complexity of a number of OWL-related languages, as well as their bias towards DL and LP semantics. The figure provides rough evidence about the expressivity of the languages, based on the complexity of entailment algorithms for them. Direct comparison between these different languages is impossible in many of the cases. For instance, Datalog is not simpler than OWL DL, it introduces a different type of complexity.

I also present on this figure the complexity supported by two of the state-of-the-art scalable repositories. Although the concrete semantics supported by those engines is not discussed here in detail, I believe it provides useful reference. The figure presents the positioning of OWLIM’s reasoning capabilities, in particular its owl-max rule-set, as defined in [20]. Based on the descriptions from [26], ORACLE 11g’s OWL Prime rule-set defines an OWL fragment similar to OWLIM’s owl-max. I have to note, however, that:

- The two languages are not the same; OWL Prime seems to be a bit simpler;
• ORACLE 11g allows usage of an external engine for TBox reasoning and materialization on the schema/ontology part of the dataset.

The results on query evaluation completeness, provided in [25], suggest that OWL Prime, combined with Pellet, [3], can answer completely all queries in the LUBM benchmark, [10]. From there I conclude that OWL Prime has complexity comparable to this of OWLIM’s owl-max rule-set, regarding ABox reasoning. The results reported in [22] suggest that DAML DB supports a language of comparable complexity as well.

An extended discussion on the subject can be found at:
3 Semantic Repository Tasks and Performance Factors

The performance of a “classical” semantic repository (SR) has to be measured at least for the following tasks:

- **Data loading**, including storing and indexing of both instance data and ontologies. The performance depends on several factors:
  - **Materialization** – whether and to what extent forward-chaining is performed at load time;
  - **Complexity of the data model** – some SR employ extended RDF data models, e.g. including named graphs. Richer data-models are more “expensive” to build and maintain.

- **Query evaluation**. There are several factors which affect the time and memory space, or more generally, the computing resources, necessary for this task:
  - **Deduction** – whether and to what extent backward-chaining is involved;
  - **Size of the result-set** – fetching large result-sets can take considerable time.
  - **Query complexity** – the number of the constraints (e.g. triple-pattern joins), the semantics of the query (e.g. negation-related clauses), the usage of operators which are tough to support through indexing (e.g. LIKE).

Searching for a meaningful measure of the evaluation speed across a set of diverse queries, I introduce a metric called query-time-per-result (QTPR). It is calculated as query evaluation time divided by the number of the results. Although QTPR is still not the perfect query performance measure, I claim that it is much more useful than the query evaluation time, taken as it is. The least to say is that it allows for accounting the fetching time, which has a serious impact for large result sets. It also allows getting a meaningful average measure across query sets where the number of results varies in orders of magnitude.

“Modifications” to the data and/or schemata, e.g. updating and deleting values, represent another important class of the tasks performed against a SR. I am not discussing modifications here, because their complexity and importance can change considerably across applications and usage patterns.

Similarly, transaction size and level of isolation supported may also have serious impact on the performance. Although many semantic repositories provide some extent of transaction isolation, it is usually less comprehensive than this of mature RDBMS. The same holds for handling large numbers of simultaneous users – while most of the SR can handle multiple users their performance under serious load is not well studied. For this reason, transaction isolation and number of users are not discussed further in the paper.
3.1 Performance Dimensions

Reasoning performance targets are defined in terms of speed, in the sense of either throughput or response time. There are several parameters which affect the speed of a semantic repository for both loading and query evaluation:

- **Scale** – the size of the repository in terms of number of facts/triples. For engines using forward-chaining and materialization, the volume of the data they have to play with includes the inferred facts/triples. Note that the RDF/OWL representation of Wordnet\(^3\) expands after materialization from 1.9 million statements to 8.2 million;

- **Reasoning complexity** – the complexity of the ontology/logical language, the specific ontology, and the dataset. E.g. a highly interconnected dataset, with long chains of transitive properties, can appear much more challenging for reasoning compared to another dataset, even when both are encoded against one and the same ontology;

- **Hardware and software setup** – the performance can vary considerably depending on the version/configuration of the compiler/virtual machine, the operating system, the configuration of the SR itself and the hardware configuration, of course.

The above analysis demonstrates that one should be realistic about the expectations regarding the utility of a simple target for the performance of a SR. Each specific benchmark provides information about the performance of an engine for a specific combination of parameters; it presents a low-dimensional section (the benchmark results) of the multi-dimensional “body” of the SR performance. A benchmark designed to measure the performance of a SR, dealing with single user’s calendar and contact data on a smart phone, is not likely to be useful for evaluating another SR, which should allow hundreds of users to investigate billions of facts in the life science domain. The same holds for the RDBMS – one can hardly define a single simple performance target to measure their performance on. It took decades for the RDBMS community to come up with standard benchmarks like TPC (http://www.tpc.org/). Yet, the major RDBMS vendors have policy to avoid competitive performance benchmarks. Today, TPC is primarily used by hardware and OS vendors to compare the performance and the cost efficiency of server configurations.

3.2 Nowadays Measuring Sticks: LUBM and UNIPROT

Here I present some benchmarks and datasets which are most commonly used as measuring sticks for the performance of semantic repositories. I refer to these benchmarks in the state of the art analysis (section 4) and in the definition of the scalability targets (section 5).

---

\(^3\) Wordnet is the most popular lexical knowledge base, http://www.w3.org/TR/wordnet-rdf/. Results on loading its RDF/OWL representation in OWLIM are presented in [20].
The Lehigh University Benchmark (LUBM) is an outstanding benchmark for OWL repository scalability, defined in [10]. It employs synthetically generated datasets using a fixed OWL ontology of university organizations. The complexity of the language constructs used is between the one of OWL Horst and OWL DLP, as shown on Figure 1. The LUBM benchmark defines 14 queries that are used to check the query evaluation speed of repositories that have loaded a given dataset. The biggest standard dataset is LUBM(50) (i.e. it contains synthetic data for 50 universities); its size is 6.8 million explicit statements, and if written to the hard disk (in 1000 RDF/XML files), it takes 600Mbytes. For the purposes of scalability measurements many groups have used the LUBM generator to create bigger datasets, e.g. LUBM(1000) and LUBM(8000) which contain respectively 130 and 1086 million explicit statements.

UOBM, [17], is a benchmark that puts on test scalable ABox (instance data) reasoning against a relatively inexpressive DL ontology. UOBM is a further development of the LUBM benchmark; it uses the same evaluation framework (i.e. Java libraries), but provides alternative ontology, knowledgebase and queries, which allow for:

- More comprehensive coverage and usage of the OWL Lite and OWL DL semantics.
- Additional connections across the datasets for different universities – this modification assures higher level of connectivity in the RDF graph, which provides a more realistic and interesting test case.

The UOBM benchmark includes two distinct datasets for OWL Lite and OWL DL, each of which includes data collections for 1, 5 or 10 universities, named respectively: Lite-1, Lite-5, Lite-10, DL-1, DL-5, and DL-10. No dataset generator for UOBM is available at present. The Lite version of the test contains 13 queries; the DL version adds two more.

UniProt\(^4\) (Universal Protein Resource) is the world's most comprehensive and most popular database of information on proteins, created by joining the information contained in several other resources (Swiss-Prot, TrEMBL, and PIR). UniProt RDF, [24], is an RDF representation of the database with respect to an OWL ontology, expressed in a sub-language of OWL Lite. As it represents one of the largest real-world datasets represented in RDF and OWL, managing UniProt it is often used as benchmark for scalability and reasoning capabilities of semantic repositories. At present, the size of the RDF representation of UNIPROT is about 384M statements. The RDF graph defined in the UNIPROT ontology is highly interconnected, which has significant impact on the reasoning speed of semantic repositories.

---

\(^4\) [www.uniprot.org/](http://www.uniprot.org/)
4 State of the Art of Scalable Reasoning

I will try to provide below the most relevant evaluation data publicly available about the state of the art of “classical” semantic repositories. Before analyzing evaluation data, it should be noted that there are very few results reported in the range of hundreds of millions of statements. Further, the published evaluation data is often incomplete, e.g. loading times are published, without data regarding query evaluation.

Based on all public results, currently the only schema for implementation of a semantic repository, which allows for sound and complete reasoning and query evaluation on top of a billion of RDF triples, is total materialization (see section 2.1). The UNIPROT results of RDF Gateway, [15], declared to be a deductive database, can be considered an exception. However, there is very limited information available about the loading (takes “several days”, without inference) and query evaluation, especially when the query requires non-trivial inference.

I will first comment on some scalability results related to OWL DL and then move towards the more elaborate discussion regarding specific semantic repositories, which can handle huge datasets with reasoning of smaller complexity.

4.1 Scalability of DL Reasoning

I am not aware of published results about practically usable setup of a semantic repository endorsing OWL-DL semantics on top of more than 10 million statements\(^5\). As mentioned in section 2.2, the fundamental reason is that the DL formalisms (including OWL Lite) are not tractable. DL results on larger datasets are reported under specific constraints:

- incomplete or approximate reasoning;
- support for concrete reasoning tasks, but not the full functionality expected by a semantic repository. Such examples are specific modes of inconsistency checking.

Below, I provide references to some of the outstanding results in this area, although they do not match the principle scalability targets set out in section 1. The purpose is to provide evidence about the current capabilities of the engines working with semantics at this class of complexity, so that they can be used as basis for outlining the corresponding targets for incomplete reasoning.

4.1.1 UOBM Benchmark Results

I have summarized the most interesting UOBM benchmark (see section 3.2) results in Table 1. The size of the respective datasets is indicated in millions of explicit statements. The information about Lite-1 and DL-1 is given in order to provide evidence how loading and

\(^5\) An exception are the results of SHER, discussed later on in this section. However, it is unclear to what extent they can be reproduced in a standard semantic repository benchmark.
query times depend on the size of the UOBM dataset for the different tools. QTPR is a metric defined in section 3.

<table>
<thead>
<tr>
<th>Table 1. Results for UOBM (Simple DL Reasoning)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OW=SwiftOWLIM MI=MINERVA</td>
</tr>
<tr>
<td>Loading time (sec)</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>20 000</td>
</tr>
<tr>
<td>Aver. Query Time (s.)</td>
</tr>
<tr>
<td>Aver. QTPR (msec.)</td>
</tr>
</tbody>
</table>

Interpreting the above results, one should consider that the results for MINERVA, [17], are measured in a setup (Pentium IV 2.66 GHz, JDK 1.4.2), which is about twice slower than the one of OWLIM (Pentium Core 2 Duo 3.00 GHz, JDK 1.6).

According to [17], MINERVA was able to load the DL-10 set and answer the first 12 queries completely, and queries 13\textsuperscript{th} and 14\textsuperscript{th} incompletely. As no concrete query evaluation times are provided, the numbers that I provide in the table are guesses based on the figures.

SwiftOWLIM (ver. 2.9.1, [19]) is able to load DL-10 on a desktop machine (Java heap maximum set to 512MB) and answer completely all the queries, except the 15\textsuperscript{th}, which require ABox reasoning not supported by SwiftOWLIM. Using an excerpt of the dataset published at IBM SHER reasoner’s page\(^6\), SwiftOWLIM scaled up to DL-20 and DL-25 in 32-bit environment. The loading speed was decreasing in sub-logarithmic fashion. Some peculiarities in this dataset, caused strange slowdowns when the complete DL-30 set is loaded. It seems to me that the UOBM dataset generator is far less predictable and transparent, compared to the one of the original LUBM.

In theory, for UOBM SwiftOWLIM should have been able to perform better in its “transitive” mode, which implements a hybrid reasoning strategy. In particular, the inferred closures of transitive, inverse, and symmetric properties are not materialized, but rather handled programatically, in a sort of backward- chaining fashion. As reported in [20], this hybrid strategy proven to be very efficient in handling purely synthetic datasets, as the one of the EXQUANT test defined in [25]. Out of a sudden, although UOBM this was not the case with UOBM – SwiftOWLIM handles it more efficiently in its standard “total materialization” mode.

Follows my analysis on the UOBM results above:

---

• The complexity of incomplete reasoning against some light-weight DL ontologies and datasets can grow in linear dependency on the size of the dataset. The loading speed decreases in sub-logarithmic fashion. The query evaluation times grow fast, but QTPR grows linearly, i.e. the major reason for the lower query times are the growing result sets;

• It is important to note that the query times are far from the constant behavior expected by a RDBMS. The explanations are different for the different reasoning strategies. In the case of SwiftOWLIM, the queries are slowed down by the partial backward-chaining performed at query/retrieval-time.

Based on simple extrapolation, I believe that a “regularly” grown DL-30, can be loaded on 32-bit Java machine (e.g. less than 1.6 GB of Java heap). Experimental data show that the speed for loading these approx. 7M statements with DL semantics will be close to 8000 statement/sec. In terms of query evaluation performance, I expect QTPR values in the range 10-20 msec. On a 64-bit server hardware (given 15 GB of Java heap) SwiftOWLIM should be able to handle an UOBM DL-250 dataset, with size between 50 and 60 million explicit statements. The query evaluation performance can be expected to be in the range of 100 msec. QTPR. Such an experiment has not been carried out because the instance data generator for UOBM is not publicly available.

4.1.2 SHER and ABox Summarization

[5] reports interesting results on the UOBM performance of SHER – a scalable high-performance reasoner developed at IBM. SHER’s scalability is related to its reasoning strategy, which can be outlined as follows:

1. The ABox is “summarized” and the resulting smaller ABox, together with the original TBox (the concept definitions), is populated in Pellet ([3], one of the outstanding “classical” DL reasoners);

2. SHER transforms the reasoning operations into inconsistency checking, that is performed by Pellet. For example, the instance retrieval query “give me all instances of concept C” can be modeled by adding to the ABox assertion “NOT x:C”, i.e. “there is no individual which belongs to C” and then checking consistency of the KB;

3. In case the summarized ABox appears consistent, then the original is also consistent, so, the according conclusions are drawn by SHER and the reasoning tasks ends;

4. If the summarized ABox appears inconsistent, SHER starts a procedure for “refinement” of the summary, and then the extended summary ABox is again checked through Pellet;

5. If the refined ABox appears consistent, SHER continues as in step 3 above. Otherwise it goes back to step 4 and continues with the refinements, i.e. with the “de-compression” of the summarized ABox, until all expansions relevant to the current task are made;
6. In case that no consistent ABox was found, SHER draws “justifications” for the inconsistency, i.e. sets of triples which render the ABox inconsistent. In the case that the original task was a particular query, these justifications represent the answers of the query. Otherwise, the justifications can be considered just as such, so, they could suggest noise or inconsistencies in the ABox.

The above schema allows scaling up particular types of inference tasks on DL ontologies considerably, as DL reasoning should be performed against the summarized ABox, which usually appears to be some orders of magnitude smaller than the original one.

[5] reports on the performance of instance retrieval queries for all 112 concepts of UOBM against the DL-30 dataset (approx. 7 million statements). The average query time measured is 35 sec. Considering that there are approx. 709 159 type assertions, it suggests QTPR in the range of 5 msec. This is a very impressive result for complete reasoning on top of such a large ABox. However, the following should be considered:

- The instance retrieval queries are simpler than the ones included in UOBM.
- The approach of query answering through inconsistency checking is expensive for application on conjunctive queries. This could be an explanation of why [5] provides no results of evaluation with the original UOBM queries.
- Loading time is not reported.

[21] reports on a case study where SHER was used to match patient records to clinical trials using the SNOMED ontology. An important aspect of this experiment was that it involves a large TBox. To handle this, SHER implements another important optimization – it excludes from the process the irrelevant part of the TBox, namely, those concepts which are not referred in the ABox and which are not related to concepts that are referred there. This operation is performed through FACT++ ([13], another outstanding DL reasoner). Originally, there were about 500 thousand assertions in the TBox, out of which only 22 561 thousand remained and were used in the course of the ABox reasoning. The dataset put on test contained approx. 60 million statements – data from clinical records of 240 thousand patients for 1 year. The results can be summarized as follows:

- Loading and TBox “minimization” times are not reported;
- Queries took between 30 and 300 minutes and return between 2 and 5555 results. The QTPR ranges between 1 and 1830 seconds, with 213 sec. average.

The results of SHER are interesting as they provide important evidence for a new approach of scaling up DL reasoning. The hardware used in the experiments is comparable to the one discussed below for the highly scalable repositories.

The approach of SHER can be compared with the incomplete reasoning scheme proposed in the LARKC project. In particular, the summarization step can be considered as a sort of abstraction, while the iterative refinement represents a kind of retrieval of an increased sample.
4.2 Systems and Datasets

I will focus on the following performance results of some of the state-of-the-art semantic repositories:

- LUBM(8000) and LUBM(1000) benchmarks of AllegroGraph, BigOWLIM, Virtuoso, ORACLE, and DAML DB;
- UNIPROT loading of AllegroGraph and BigOWLIM.

Table 2 presents details about the tools and configurations behind the reported results.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Version</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWLIM</td>
<td>BigOWLIM v.0.9.6</td>
<td>Server with two Opteron 270 (2GHz, dc) CPUs, 16GB of RAM, four SATA drives in RAID 10</td>
</tr>
<tr>
<td>AllegroGraph [7][8]</td>
<td>AllegroGraph 64-bit RDFStore, ver. 2.0 and 2.2</td>
<td>Server with two Opteron 844 (1.8 GHz, sc) CPUs and 16 GB of RAM</td>
</tr>
<tr>
<td>Virtuoso</td>
<td>Open source version 4.5, rev. 5, of Virtuoso server</td>
<td>Server with two Xeon 5130 (2 GHz, dc) CPUs, 8GB RAM, four SATA drives</td>
</tr>
<tr>
<td>ORACLE</td>
<td>Early release of 11g</td>
<td>PC with 3GHz CPU (?), 2GB of RAM, two SATA drives, using two PCs for storage over Gigabit Ethernet (6 HDD in total)</td>
</tr>
<tr>
<td>DAML DB</td>
<td>Ver. 2.2.1.2, via Sesame ver. 1.2.6</td>
<td>Server with two Xeon E5345 CPUs (2.33 GHz, qc), 16 GB RAM, 6 TB storage (probably RAID over 10+ drives)</td>
</tr>
</tbody>
</table>

The server configurations used for the runs of the different systems seem comparable, keeping in mind the following notes:

- ORACLE was benchmarked on a desktop machine with less memory; on the other hand [25] reports data on inference only.
- The server used for the DAML DB runs, see [22], seems more powerful than the others referred here. In terms of CPU power, the machine should be considered comparable to the others, since, to the best of my knowledge, the engines included and the benchmark setup benefit little from more than two CPU cores. Although there is no concrete information in [22], the storage system is probably much faster than the ones used in the runs that we discuss below.

In addition, [22] provides no concrete measurement data, but only diagrams where load and query times are presented on a logarithmic scale, with respect to the growth of the repository size. In order to be able to position it properly on my diagrams below, I made imprecise approximations for the figures behind the diagrams. I would be happy to update this document with more concrete data, shall such be provided to me.
In general, it is important to note that the reader should consider the data below as sample figures, which provide evidence about the current state of the art in scalable reasoning, rather than as comparative analysis between the tools.

Results related to indexing and querying 7 billion statements in cluster of 16 machines are published for YARS2, [11]. There is no information about the load time and no inference takes place. The query evaluation times indicate average values in the range of hundreds of milliseconds. However, they cover only simple queries with one or two joins. For these reasons, the above mentioned experiments with YARS2 are not discussed below.

4.3 Loading Performance

Let me start with the loading performance. For the engines and experimental setups listed here, loading involves:

- Parsing of the input RDF;
- Persisting the data and indexing (including the implicit statements inferred);
- Inference, namely materialization through forward-chaining (OWLIM and ORACLE).

The times provided for ORACLE include only inference, persistence and indexing of the implicit statements; the explicit ones were loaded in advance.

Table 3 presents the loading time and speed of the engines for the different datasets. For each run I provide the following data:

- **Scale** and **Time**, in number of explicit statements loaded and hours, respectively;
- **Speed**: average speed of loading, in thousands of statements per second;
- **Overall complexity index**: average of the Forward-Chaining and the Parsing Complexity Indices;
- **Forward-chaining semantics**: what language is used for forward-chaining and materialization during the loading, if any;
- **Forward-chaining complexity-index**: my subjective estimate about the complexity of the reasoning during this run;
- **Parsing & Indexing Complexity**: my subjective estimate about the complexity of the loading tasks that are not related to indexing. It includes the specifics of the concrete run and system, e.g.: rich RDF model, full-text search indexing, etc.
Table 3. Loading Performance

<table>
<thead>
<tr>
<th>Semantic Repository</th>
<th>Dataset / Run</th>
<th>Scale (mill. ex.st.)</th>
<th>Time (hours)</th>
<th>Speed (KSL/sec.)</th>
<th>Overall Compl. Index</th>
<th>Forward-Chaining Semantics</th>
<th>Forw.Ch. Compl. Index</th>
<th>Pars.&amp; Index. Compl.</th>
<th>Rich RDF Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>BigOWLIM</td>
<td>LUBM(8000,0)</td>
<td>1 068</td>
<td>34.01</td>
<td>8 724</td>
<td>14</td>
<td>OWL-Horst</td>
<td>20</td>
<td>7</td>
<td>-</td>
</tr>
<tr>
<td>BigOWLIM</td>
<td>LUBM(8000,0)</td>
<td>1 068</td>
<td>14.43</td>
<td>20 565</td>
<td>9</td>
<td>RDFS</td>
<td>10</td>
<td>7</td>
<td>-</td>
</tr>
<tr>
<td>BigOWLIM</td>
<td>LUBM(1000,0)</td>
<td>138</td>
<td>2.46</td>
<td>15 602</td>
<td>14</td>
<td>OWL-Horst</td>
<td>20</td>
<td>7</td>
<td>-</td>
</tr>
<tr>
<td>BigOWLIM</td>
<td>LUBM(1000,0)</td>
<td>138</td>
<td>0.96</td>
<td>40 070</td>
<td>9</td>
<td>RDFS</td>
<td>10</td>
<td>7</td>
<td>-</td>
</tr>
<tr>
<td>BigOWLIM</td>
<td>UNIPROT</td>
<td>383</td>
<td>23.27</td>
<td>4 572</td>
<td>29</td>
<td>OWL-Max</td>
<td>50</td>
<td>7</td>
<td>-</td>
</tr>
<tr>
<td>BigOWLIM</td>
<td>UNIPROT</td>
<td>383</td>
<td>7.46</td>
<td>14 263</td>
<td>16</td>
<td>RDFS+</td>
<td>25</td>
<td>7</td>
<td>-</td>
</tr>
<tr>
<td>AllegroGraph</td>
<td>LUBM(8000,0)</td>
<td>1 107</td>
<td>23.50</td>
<td>13 085</td>
<td>6</td>
<td>Type-cache</td>
<td>2</td>
<td>10</td>
<td>+</td>
</tr>
<tr>
<td>AllegroGraph</td>
<td>LUBM(1000,0)</td>
<td>138</td>
<td>2.44</td>
<td>15 710</td>
<td>6</td>
<td>Type-cache</td>
<td>2</td>
<td>10</td>
<td>+</td>
</tr>
<tr>
<td>AllegroGraph</td>
<td>UNIPROT</td>
<td>234</td>
<td>3.35</td>
<td>19 403</td>
<td>6</td>
<td>Type-cache</td>
<td>2</td>
<td>10</td>
<td>+</td>
</tr>
<tr>
<td>Virtuoso</td>
<td>LUBM(8000,0)</td>
<td>1 068</td>
<td>23.45</td>
<td>12 651</td>
<td>3</td>
<td>None</td>
<td>0</td>
<td>5</td>
<td>+</td>
</tr>
<tr>
<td>ORACLE</td>
<td>LUBM(1000,0)</td>
<td>138</td>
<td>7.20</td>
<td>5 324</td>
<td>12</td>
<td>OWL-Prime + Pellet</td>
<td>20</td>
<td>3</td>
<td>+</td>
</tr>
<tr>
<td>DAML DB</td>
<td>LUBM(8000,0)</td>
<td>850</td>
<td>23.00</td>
<td>10 266</td>
<td>14</td>
<td>RDFS+</td>
<td>20</td>
<td>7</td>
<td>-</td>
</tr>
<tr>
<td>DAML DB</td>
<td>LUBM(8000,0)</td>
<td>220</td>
<td>10.00</td>
<td>6 111</td>
<td>14</td>
<td>RDFS+</td>
<td>20</td>
<td>7</td>
<td>-</td>
</tr>
</tbody>
</table>

The result of BigOWLIM for UNIPROT, marked with inference complexity index 25 (RDFS+), corresponds to a run of BigOWLIM with its RDFS rule-set. BigOWLIM has built-in support for owl:sameAs equivalence, which is always active, disregarding the result set selected. As UNIPROT makes heavy usage of owl:sameAs, this leads to higher complexity of the forward-chaining process.

AllegroGraph’s UNIPROT run uses an older and smaller version of the dataset.

There was no sufficient information publicly available to allow me to make proper estimation of the different types of loading complexity for DAML DB. Based on the available information, the complexity should be comparable to the one faced by BigOWLIM when loading LUBM. Thus, I have approximated it to be equal to the corresponding complexities for BigOWLIM. Shall more concrete data become available to me, I will be happy to update the figures.

Figure 2 represents in a graphical way the dependencies between the sizes of the datasets, the loading complexity of the run, and the loading speed of the different engines.
One can see that the results for loading are comparable, taking into account that the engines differ in features. Taking BigOWLIM’s results from LUBM(1000), one can observe how the difference in the semantics supported can alter the loading time almost three times.

### 4.4 Query Evaluation Performance

The second major performance criteria that I analyze here is query evaluation speed. There is very little public information available about query performance on datasets with close to one billion statements:

- The YARS2 data in [11] – they were excluded for the reasons given above;
- The results of BigOWLIM, evaluating the LUBM(8000) queries in [19].
- The results of DAML DB for the LUBM(8000), published in [22] – those are relevant, but they lack concrete numbers, rather provide graphic on a logarithmic scale instead.

In Table 4 I present results published in [19], which compare the query evaluation times of two versions of BigOWLIM (0.9.5 and 0.9.6). Table 4 presents the number of results
returned for each of the queries and the query evaluation times of each version for each of the queries. We can see that the number of the results returned varies between 83 millions and 4, i.e. there is a difference of some 7 orders of magnitude. Obviously, the criteria for fast evaluation between queries number 1 and 6 are different. For this purpose I adapt the QTPR (query-time-per-result) metric introduced in section 3.

<table>
<thead>
<tr>
<th>Query No</th>
<th>Number of results</th>
<th>BigOWLIM 0.9.5</th>
<th>BigOWLIM 0.9.6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time (ms)</td>
<td>QTPR</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>3 962</td>
<td>991</td>
</tr>
<tr>
<td>2</td>
<td>2 528</td>
<td>1 144 726</td>
<td>453</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>24 506</td>
<td>4 084</td>
</tr>
<tr>
<td>4</td>
<td>34</td>
<td>123 978</td>
<td>3 646</td>
</tr>
<tr>
<td>5</td>
<td>719</td>
<td>31 018</td>
<td>43</td>
</tr>
<tr>
<td>6</td>
<td>83 557 706</td>
<td>169 375</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>67</td>
<td>123 806</td>
<td>1 848</td>
</tr>
<tr>
<td>8</td>
<td>7 790</td>
<td>42 514</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>2 178 420</td>
<td>2 318 412</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>37 704</td>
<td>9 426</td>
</tr>
<tr>
<td>11</td>
<td>224</td>
<td>413</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>15</td>
<td>1 317</td>
<td>88</td>
</tr>
<tr>
<td>13</td>
<td>37 118</td>
<td>22 536</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>63 400 587</td>
<td>383 281</td>
<td>0</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>1471</td>
<td></td>
</tr>
</tbody>
</table>

The average QTPR of BigOWLIM ver. 0.9.6 on LUBM for dataset of one billion statements is about 13 milliseconds. The result of the older version (0.9.5) of the same engine is over a hundred times lower. The major difference between the two versions is that the newer one makes better use of statistics for the sake of query plan optimizations, similar to those employed in the RDBMS. The difference in the results demonstrates an expected fact, namely, that query optimization is crucial for datasets of this size. As this type of query optimizations are practically impossible for a reasoning engine, which performs backward-chaining in the process of query evaluation, the above result provides evidence that such engines will face principle problems with their query evaluation performance on large datasets.

In my interpretation the query evaluation results given in [22] suggest the following query times of DAML DB over a 850 million statements subset of LUBM(8000):

- Query 1 time: ~1 hour; QTPR: 15 min. Obviously DAML DB’s performance here can be a subject of improvement.
• Query 14 time: ~6 hours; QTPR ~0.3 msec. Although this is much slower than the 83 sec. needed by BigOWLIM, the QTPR is good;
• Query 9 time: ~1 hour; QTPR ~2 msec. Again, the QTPR is good.

Considering that the current version of BigOWLIM works with “simple” RDF model (e.g. without support for Named Graphs), one can expect that a well tuned engine can achieve at present speeds in the range of 10-30 milliseconds QTPR on LUBM(8000), depending on its other features.
5 Performance Targets

This section introduces measurable performance targets for semantics repositories, based on the discussion on the relevant tasks and factors (section 3), and on the analysis of the current state of these systems (section 4). My aim is to provide measuring sticks, which could facilitate the development of this type of technology over the next couple of years.

I define four semantic repository performance targets (SRPT) as follows:

**SRPT1: Complete LUBM reasoning.** LUBM(8000) should be loaded and its queries shall be answered in sound and complete fashion. LUBM requires light-weight, tractable OWL reasoning. The dataset allows for easy partitioning into isolated sub-graphs. **Target:** loading speed 100,000 explicit st./sec.; average QTPR below 4 msec.

**SRPT2: Complete UNIPROT reasoning.** The UNIPROT dataset should be loaded; the forward chaining complexity of UNIPROT is higher than this of LUBM. Queries comparable in complexity to the set published at RDF Gateway’s UNIPROT experiments shall be evaluated; the queries shall involve the LIKE operator or similar. **Target:** loading speed 20,000 explicit st./sec.; average QTPR below 300 msec.

**SRPT3: Incomplete OWL DL ABox reasoning.** A dataset corresponding to the DL version of the UOBM benchmark for 500 universities should be generated (approx. 110M st.) The corresponding queries should be answered in incomplete fashion while the following IR-style accuracy measures should be met: precision 95% at recall 25%. **Target:** loading speed 50,000 explicit st./sec.; average QTPR below 30 msec.

**SRPT4: Incomplete UNIPROT reasoning.** As SRPT2, but with incomplete query answering; the following accuracy measures should be met: precision 98% at recall 10%. **Target:** loading speed 50,000 explicit st./sec.; average QTPR below 100 msec.

When using the above-mentioned performance targets, the following should be considered:

- The targets shall be met on a single “general purpose” server with cost up to 10,000 EURO by the end of 2010. Moor’s law is likely to stultify them in a longer term;
- Adjustments may be necessary if tests are made with version of UNIPROT which differs in size or complexity from the version available in the fall of 2007;
- Relevance ranking is not properly covered by the above specified targets.

I do not define specific targets for distributed semantic repositories. Since at present distributed implementations cannot beat the price/performance offered by single-server installations, the above goals should provide good test for distributed implementations. In a distributed set up, the price constraint should apply to the cost of the set of the machines, including the network/connectivity hardware.

The scalability targets defined here are just four points in the multidimensional space of the semantic repository performance. Thus, they do not define well-founded criteria for evaluation of such engines. They are rather meant as buoys that could help the navigation in the scalable reasoning area.
6 Conclusion

It is important to note that the goals of both projects which inspired this paper, namely LARKC and RASCALLI, are not to improve the performance of the “classical” reasoning engines. The primary target in LARKC is a new reasoning paradigm, a new definition of the tasks to be performed and objectives to be met by a semantic repository. In RASCALLI scalable reasoning is a secondary target and the objectives go in the direction of cognitive models far beyond the dangerous convenience of the classical mathematical logic.

Still, based on the state of the art analysis and the above-defined performance targets, Table 5 provides estimates about the expected performance and scalability advances.

<table>
<thead>
<tr>
<th></th>
<th>2007 SOA</th>
<th>2010 Targets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum scale</td>
<td>1 billion statements</td>
<td>20 B st. (complete)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100 B st. (incomplete)</td>
</tr>
<tr>
<td>Speed of loading 1B statements</td>
<td>2 000-20 000 st./sec.</td>
<td>10 000-100 000 st./sec</td>
</tr>
<tr>
<td>Query evaluation against 1B statements (average QTPR)</td>
<td>10 msec.</td>
<td>4 msec. (complete)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 msec. (incomplete)</td>
</tr>
<tr>
<td>Relevance ranking</td>
<td>Not present</td>
<td>Present</td>
</tr>
<tr>
<td>Inference strategy control</td>
<td>None/heavy</td>
<td>Plug-able inference modules</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cost/benefit-based control</td>
</tr>
</tbody>
</table>

Extending my initial comments, it should be realized that there is limited utility in comparing the performance of the technology that LARKC aims at against the performance of today’s engines. I would make an analogy with a comparison between the advances made by the early aircrafts and sea ships, based on measurements with a lead.

An earlier version of this text has been included in the LARKC project documentation.

I want to thank professors Dieter Fensel and Frank van Harmelen for their comments and encouragements to publish this document, as well as my colleagues Vassil Momtchev and Borislav Popov for their corrections and suggestions. I shall express special gratitude to Zlatina Marinova, who took the burden of editing the manuscript, bringing it in a much more readable and consistent shape. Finally, the semantic repositories landscape would have been a much different place without the work of Damyan Ognyanoff and Ruslan Velkov on OWLIM. I believe that without their efforts, the overall level of ambition would have been much lower.
7 References


