



LarKC

The Large Knowledge Collider:

a platform for large scale integrated reasoning and Web-search

FP7 – 215535

D5.5.2 Validation goals and metrics for the LarKC platform

Coordinator: Atanas Kiryakov, Onto

With contributions from: Zdravko Tashev, Damyan Ognyanoff, Ruslan Velkov, Vassil Momtchev, Boiko Balev, Ivan Peikov

Quality Assessor: Eyal Oren, Stefan Wesner, Michael Witbrock

Quality Controller: Georgina Gallizo

Document Identifier:	LarKC/2008/D5.5.2 /v1.1
Class Deliverable:	LarKC EU-IST-2008-215535
Version:	1.1
Date:	31.08.2009
State:	Final
Distribution:	Public



EXECUTIVE SUMMARY

This deliverable is dedicated to validation of the scalability and the performance of the data layer of the LarkC platform. Its general purpose is to set validation targets for the development of the data layer. As part of the analysis of the state of the art, it also provides concrete measurement results about OWLIM – the engine in the basis of the current implementation of the data layer. Other wide-ranging methods for measuring the performance of other components such as individual plug-ins, as well as the design of measures for the performance of the platform as a whole will be discussed in D1.4.1 “Initial framework for measuring and evaluating heuristic problem solving” (due M18).

The data layer represents a basic component of the platform, which supports all plug-ins and applications with respect to storage, retrieval, and light-weight inference on top of large volumes of data. It supports a range of data-access modalities, e.g. Java API, SPARQL endpoint and linked data publishing. The framework for its validation and evaluation is defined in terms of tasks and aspects that need to be investigated. The major tasks are loading, inference, and query evaluation. The major performance factors are scale, reasoning complexity, number of clients, and software and hardware environment.

The experiments reported here involve the most popular benchmarks (e.g. LUBM and BSBM), as well as evaluation of the data layer against LDSR and PIKB – two datasets that can be considered as reason-able views to the web of linked data. LDSR is huge knowledge base of common facts combining DBPedia, Geonames, Wordnet, and other datasets. It is used for evaluating the selection plug-ins and ranking schemata in WP2 “Retrieval and Selection”. PIKB is the dataset used in the two life science use cases (WP7a “Early Clinical Drug Development” and WP7b “Carcinogenesis Research”). It reflects the complexity and the specifics of the data integration problems in this domain.

Overall, the evaluation provides evidence that the implementation of LarkC platform’s data layer is very well positioned with respect to the other outstanding engines in the highly competitive niche of the so-called semantic repositories. The scalability of OWLIM matches the requirement of the use cases at the current stage of the project. For instance, it can load and manage all datasets of the Linking Open Data project within a single commodity database server. There are no published results for other tools that can demonstrate similar scale for lightweight inference.

In order to reach the next level of scalability, LarkC’s data layer will have to provide efficient means for distributed data management, spanning to tens and hundreds of machines. This requires advancing the frontiers of the state of the art in distributed RDF database management, where the best current results involve several machines and report scalability and performance results that are easy to match with a single machine at a fraction of the cost.



DOCUMENT INFORMATION

IST Project Number	FP7 - 215535	Acronym	LarKC
Full Title	The Large Knowledge Collider: a platform for large scale integrated reasoning and Web-search		
Project URL	http://www.larkc.eu/		
Document URL			
EU Project Officer	Stefano Bertolo		

Deliverable	Number	D5.5.2	Title	Validation goals and metrics for the LarKC platform
Work Package	Number	WP5	Title	The Collider Platform

Date of Delivery	Contractual	M14	Actual	M15
Status	version 1.1		final x	
Nature	prototype <input type="checkbox"/> report x dissemination <input type="checkbox"/> other <input type="checkbox"/>			
Dissemination level	public x consortium <input type="checkbox"/>			

Authors (Partner)				
Responsible Author	Name	Atanas Kiryakov	E-mail	naso@sirma.bg
	Partner	Onto	Phone	

Abstract (for dissemination)	<p>This deliverable is dedicated to validation of the performance and scalability of LarKC platform's data layer. The framework for validation is defined in terms of tasks (loading, inference, query evaluation) and aspects (scale, reasoning complexity, number of clients) to be investigated. Specific targets are set on the basis of analysis of the state of the art in the field of scalable RDF databases and light-weight inference. The data layer is evaluated against LDSR and PIKB – two datasets that represent “reason-able views” to the web of linked data, already used in the project. Query evaluation performance is benchmarked against the Berlin SPARQL Benchmark (BSBM). The results show that the data layer is well positioned among today's best semantic repositories. In order to reach the next level of scalability, the data layer of LarKC will have to provide efficient means for distributed data management, spanning to tens</p>
-------------------------------------	--












	and hundreds of machines.
Keywords	Validation, evaluation, scale, performance, query evaluation, semantic repository, lightweight reasoning

Version Log			
Issue Date	Rev. No.	Author	Change
19/06/09	0.9	Atanas Kiryakov	Updates in section 3, regarding Eyal's comments and others. Update in the tables and figures on loading in section 4.2.
23/07/09	0.97/20	Ivan Peikov, Atanas Kiryakov	Finalized up to section 4.3, inclusive. Major updates in the LDSR section and in section 4.2 (the descriptions of the engines). Section 4 restructured – sub-section 4.1 added to describe the benchmarks used. Updates were made also in section 2 (validation framework); 2.4 “full-cycle benchmarking” added
23/07/09	1.0	Vassil Momtchev & Gergana Petkova	Proof-reading and QA comments implementation
31/08/09	1.1	Vassil Momtchev	1 st LarKC review comments: <ul style="list-style-type: none"> • Fixed reference errors • Corrected the meaning of “insensitive” to “intensive” • A more elaborated explanation for the discarded categories from DBPedia is added • Reworded the claims of the approaches that might load billions of statements in complete and sound way.



PROJECT CONSORTIUM INFORMATION

Participant's name	Partner	Contact
Semantic Technology Institute Innsbruck, Universitaet Innsbruck	 	Prof. Dr. Dieter Fensel, Semantic Technology Institute (STI), universitaet Innsbruck, Innsbruck, Austria, E-mail: dieter.fensel@sti-innsbruck.at
AstraZeneca AB		Bosse Andersson AstraZeneca Lund, Sweden Email: bo.h.andersson@astrazeneca.com
CEFRIEL - SOCIETA CONSORTILE A RESPONSABILITA LIMITATA		Emanuele Della Valle, CEFRIEL - SOCIETA CONSORTILE A RESPONSABILITA LIMITATA, Milano, Italy, Email: emanuele.dellavalle@cefriel.it
CYCORP, RAZISKOVANJE IN EKSPERIMENTALNI RAZVOJ D.O.O.		Michael Witbrock, CYCORP, RAZISKOVANJE IN EKSPERIMENTALNI RAZVOJ D.O.O., Ljubljana, Slovenia, Email: witbrock@cyc.com
Höchstleistungsrechenzentrum, Universitaet Stuttgart		Georgina Gallizo, Höchstleistungsrechenzentrum, Universitaet Stuttgart, Stuttgart, Germany, Email: gallizo@hlrs.de
MAX-PLANCK GESELLSCHAFT ZUR FOERDERUNG DER WISSENSCHAFTEN E.V.		Dr. Lael Schooler Max-Planck-Institut für Bildungsforschung Berlin, Germany Email: schooler@mpib-berlin.mpg.de
Ototext Lab, Sirma Group Corp		Atanas Kiryakov, Ototext Lab, Sofia, Bulgaria Email: atanas.kiryakov@sirma.bg
SALTLUX INC.		Kono Kim, SALTLUX INC, Seoul, Korea, Email: kono@saltlux.com
SIEMENS AKTIENGESELLSCHAFT		Dr. Volker Tresp, SIEMENS AKTIENGESELLSCHAFT, Muenchen, Germany, E-mail: volker.tresp@siemens.com
THE UNIVERSITY OF SHEFFIELD		Prof. Dr. Hamish Cunningham, THE UNIVERSITY OF SHEFFIELD Sheffield, UK, Email: h.cunningham@dcs.shef.ac.uk



<p>VRIJE UNIVERSITEIT AMSTERDAM</p>		<p>Prof. Dr. Frank van Harmelen, VRIJE UNIVERSITEIT AMSTERDAM, Amsterdam, Netherlands, Email: Frank.van.Harmelen@cs.vu.nl</p>
<p>THE INTERNATIONAL WIC INSTITUTE, BEIJING UNIVERSITY OF TECHNOLOGY</p>		<p>Prof. Dr. Ning Zhong, THE INTERNATIONAL WIC INSTITUTE, Mabeshi, Japan, Email: zhong@maebashi-it.ac.jp</p>
<p>INTERNATIONAL AGENCY FOR RESEARCH ON CANCER</p>	 <p>International Agency for Research on Ca Centre International de Recherche sur le Ca</p>	<p>Dr. Paul Brennan, INTERNATIONAL AGENCY FOR RESEARCH ON CANCER, Lyon, France, Email: brennan@iarc.fr</p>



TABLE OF CONTENTS

LIST OF FIGURES	9
LIST OF TABLES	10
LIST OF ACRONYMS	11
1 INTRODUCTION.....	12
1.1 SCOPE AND RELATIONS TO OTHER DOCUMENTS.....	12
1.2 ARCHITECTURAL OUTLINE.....	13
1.3 LINKED DATA.....	15
2 DATA LAYER VALIDATION FRAMEWORK	18
2.1 TASKS.....	18
2.2 PERFORMANCE FACTORS	18
2.3 PERFORMANCE DIMENSIONS	19
2.4 FULL-CYCLE BENCHMARKING	20
3 LDSR AS TEST DATASET FOR SCALABLE INFERENCE	22
3.1 REASON-ABLE VIEW TO THE WEB OF LINKED DATA.....	23
3.2 LOADING AND INFERENCE STATISTICS	25
3.3 LOADING AND INITIAL INFERENCE AGAINST LDSR	25
3.4 DBPEDIA CATEGORY HIERARCHY REFINEMENTS	26
3.5 OWL:SAMEAS OPTIMISATIONS	28
4 STATE OF THE ART IN SEMANTIC REPOSITORIES	30
4.1 BENCHMARKS AND DATASETS	30
4.1.1 <i>Lehigh University Benchmark (LUBM)</i>	30
4.1.2 <i>UniProt</i>	31
4.1.3 <i>DBPedia and Other LOD Datasets</i>	31
4.1.4 <i>Berlin SPARQL Benchmark (BSBM)</i>	31
4.2 ENGINES.....	32
4.2.1 <i>Sesame</i>	32
4.2.2 <i>4store</i>	32
4.2.3 <i>AllegroGraph</i>	33
4.2.4 <i>BigData</i>	34
4.2.5 <i>BigOWLIM</i>	34

4.2.6	DAML DB.....	36
4.2.7	Jena TDB	36
4.2.8	ORACLE	37
4.2.9	Virtuoso.....	38
4.2.10	YARS2.....	39
4.3	BENCHMARK CONFIGURATIONS AND ENVIRONMENTS	40
4.4	LOADING PERFORMANCE.....	41
4.5	QUERY EVALUATION	44
4.5.1	BSBM Results	44
4.5.2	LUBM Results.....	46
5	VALIDATION TARGETS AND FUTURE WORK	47
5.1	PERFORMANCE TARGETS	48
5.2	DATA LAYER DISTRIBUTION CONSIDERATIONS.....	49
6	CONCLUSION	51



List of Figures

FIGURE 1.LARKC ARCHITECTURE OVERVIEW.....	13
FIGURE 2.LARKC DATA LAYER POSITIONING AND ARCHITECTURE	15
FIGURE 3.MAP OF THE DATASETS IN LINKING OPEN DATA.....	16
FIGURE 4.SCALABLE INFERENCE MAP – UP TO 2MST.....	43
FIGURE 5.SCALABLE INFERENCE MAP – SCALE-OUT.....	44
FIGURE 6.BSBM QUERY RESULTS.....	45



List of Tables

TABLE 1.LDSR DATASETS.....	25
TABLE 2.SYSTEMS AND CONFIGURATIONS	40
TABLE 3.LOADING PERFORMANCE	42
TABLE 4.QUERY PERFORMANCE LUBM(8000), 1 BILLION STATEMENTS	45



List of Acronyms

BSBM	Berlin SPARQL benchmark
BSt	Billions of RDF statements (triples)
KSt/sec.	Thousands of statements per second; usually used to measure loading speed
LDSR	Linked Data Semantic Repository
LOD	Linked Open Data SWEO community project of W3C
LUBM	Lehigh University Benchmark
MSt	Millions of RDF statements (triples); explicit statements are considered if there are no other specific comments
NIS	Number of inserted statements
NSS	Number of stored statements
NRS	Number of retrievable statements
OWL	Web Ontology Language
SR	Semantic Repository
PIKB	Pathway and Interaction Knowledge Base
QTPR	Query time per second
QMPH	Query mixes per hour
RDF	Resource Description Framework
RDFS	RDF Schema specification language



1 Introduction

This deliverable is dedicated to validation of the scalability and the performance of the data layer of the LarkC platform. The latter represents a core component of the platform, which supports all plug-ins and applications with respect to storage, retrieval, and light-weight inference on top of large volumes of data. It supports a range of data-access modalities, e.g. Java API, SPARQL endpoint and linked data publishing. The standard back-end option of the data layer is the TRREE engine, but its design allows for easy integration of other engines as well.

The framework for validation and evaluation of the data layer is defined in terms of tasks and aspects that need to be investigated (section 2). Specific targets are set (in section 5) on the basis of analysis of the current state of the art in the field of scalable RDF databases and light-weight inference (section 4).

Specific evaluation work is also reported here, namely evaluation of the data layer against LDSR and PIKB – two datasets that can be considered as reason-able views to the web of linked data. LDSR is huge knowledge base of common facts that emerged as testbed for evaluation of the selection plug-ins and ranking schemata in WP2. Its current state is documented in section 3. PIKB is in the basis of the two life science use cases (WP7a and WP7b) and reflects the complexity and the specifics of the data integration problems in this domain. This dataset is presented in [2]. To validate the query performance with respect to generally recognized benchmarks, we have adopted the Berlin SPARQL Benchmark (BSBM).

The remainder of this section provides references to few related deliverables, presents outline of the LarkC platform and the role of the data layer in it, followed by a quick introduction to the linked data principles.

1.1 *Scope and Relations to Other Documents*

The general purpose of this deliverable is to set validation targets for the development of the data layer of the LarkC platform. As part of the analysis of the state of the art, it also provides concrete measurement results about OWLIM – the engine in the basis of the current implementation of the data layer.

More wide-ranging methods for evaluation of other components, such as individual plug-ins, as well as the design of measures for the performance of the platform as a whole will be discussed in D1.4.1 “Initial framework for measuring and evaluating heuristic problem solving” (due M18).

An earlier version of the validation framework for the data layer and the state of the art analysis for scalable semantic repositories was presented in D5.5.1 “Measurable Targets for Scalable Reasoning”, [26].

The impact of lightweight inference and materialization to the loading performance is systematically investigated in D1.1.2 “Empirical Analysis of the Initial Knowledge Representation Formalism”, [17]. It provides basis for better estimating the loading complexity as compared to the loading performance measurements discussed in section 4.1.

Deliverable D5.2.1 “Rapid Prototype of the LarkC”, [24], introduces the overall concept of the LarkC platform and describes the architecture of the prototype. The deliverable is useful to understand the proper place of the data layer within the entire LarkC architecture.

The Linked Data Semantic Repository (LDSR) was developed for the sake of evaluation of components related to selection and retrieval and presented in D2.4.1 “Spreading Activation Components”, [40]. Here we extend it and adopt it for more general evaluation purposes.

Further performance evaluation efforts should document the used datasets and the results according to the provisions of D6.2 “Templates of periodic report on data and performances”, [25], and as an extension of the datasets already documented in D6.4 “1st periodic report on data and performances”, [11].

Deliverable D7a.1.1 “LarkC Requirements summary and data repository”, [2], specifies “Early Clinical Development” use case requirements and identifies concrete biomedical datasets. PIKB is an information source developed in the context of this use case and used to integrate large sets of information and answer complex researcher’s questions.

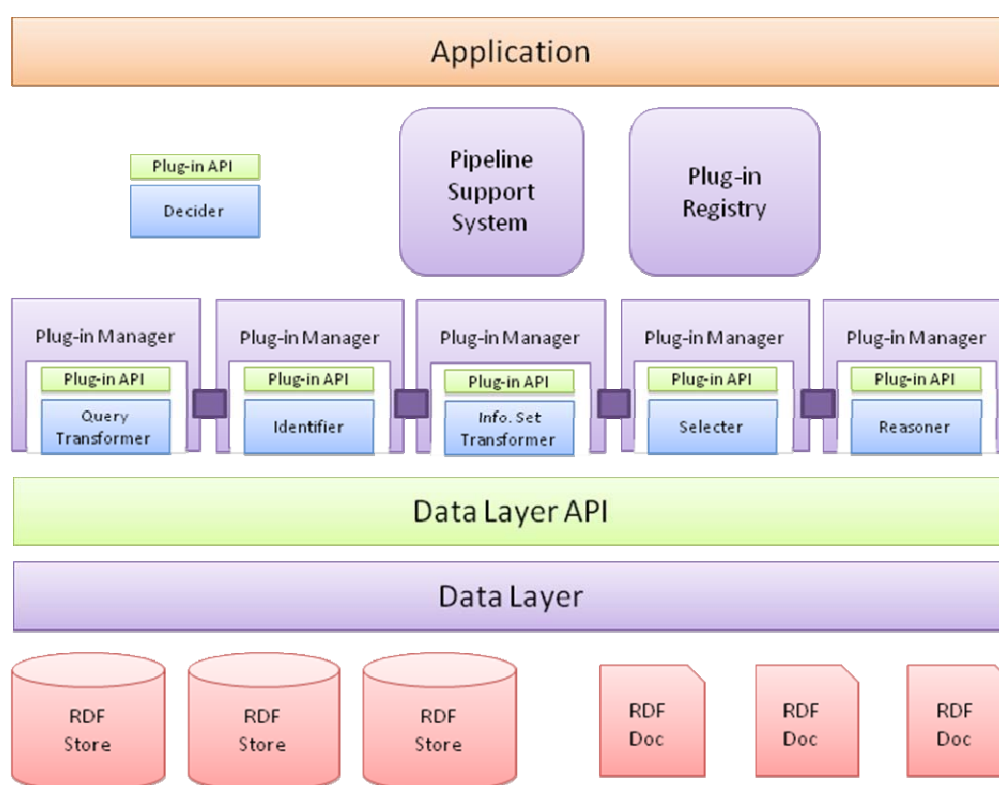


Figure 1. LarkC architecture overview

1.2 Architectural Outline

The LarkC platform aims at supporting massive distributed incomplete reasoning at web-scale. In order to simplify the development process and to maximise the reuse of software components, a plug-in framework approach is implemented. The platform serves functionality to construct, configure, and invoke different plug-ins upon request from the



Decider, which is responsible for orchestrating the complete process (see [24]). The selection plug-in plays a role in narrowing the used dataset by applying various heuristics, statistics, or other strategies. From plug-in perspective the LarKC platform is composed by:

- Plug-in API responsible for defining the contract between the different components;
- Data Layer API allowing the plug-ins to use the basic services of the data layer (storage, querying, and lightweight inference) and specifying several mechanisms to exchange RDF data between the plug-in instances;
- Plug-in Registry and Manager automating the invocation and the instantiation of the configured platform plug-ins.

The Data Layer API makes the RDF data transparent for the plug-ins and automates the passing of RDF data by reference and value. Currently, there are four different ways to pass RDF statements:

- By value – if necessary, all the statements are serialized and transferred between the plug-ins;
- By dataset reference – only a pointer to SPARQL endpoint and dataset is used to identify the set of data;
- By tripliset (labelled group) reference – ORDI framework introduces a rich RDF data model that allows to group arbitrary statements (possibly located in multiple datasets) in a named collection, called tripliset (as introduced in section 1.1 of [9] and in section 5.1 of [24]).
- By URI reference – the URI is resolved by HTTP 303 redirect and then downloaded, after the linked data principles discussed in section 1.3.

The plug-in approach guarantees a high degree of platform flexibility, simpler implementation process, and looser component coupling. However, this could have negative consequences to the system performance. This is especially true for the input/output intensive components like the Selector – a plug-in that receives as input RDF data from the local repository and a query and returns a reduced RDF set of statements, relevant to the query. In order to select a reduced set of data, the plug-in has to process huge amounts of information and, in cases of distributed deployed environments, it may take significant input/output time. To guarantee a decent performance and efficiency, an approach similar to the stored procedures in the relational database management systems is undertaken. The computation of I/O intensive operations is pushed-down in the data processing components. To keep the LarKC platform compatible with the standard SPARQL specification, new specialized predicates (instead of extended functions) are introduced to manage this stored-procedure-like mechanisms, as discussed in [40]. Figure 2 displays the implementation, as part of the data layer, of two components used for the selection:

- DualRDF: implements RDF priming;
- RDFPageRank: implements RDF rank calculation.

The implementations of both components are presented in [40].

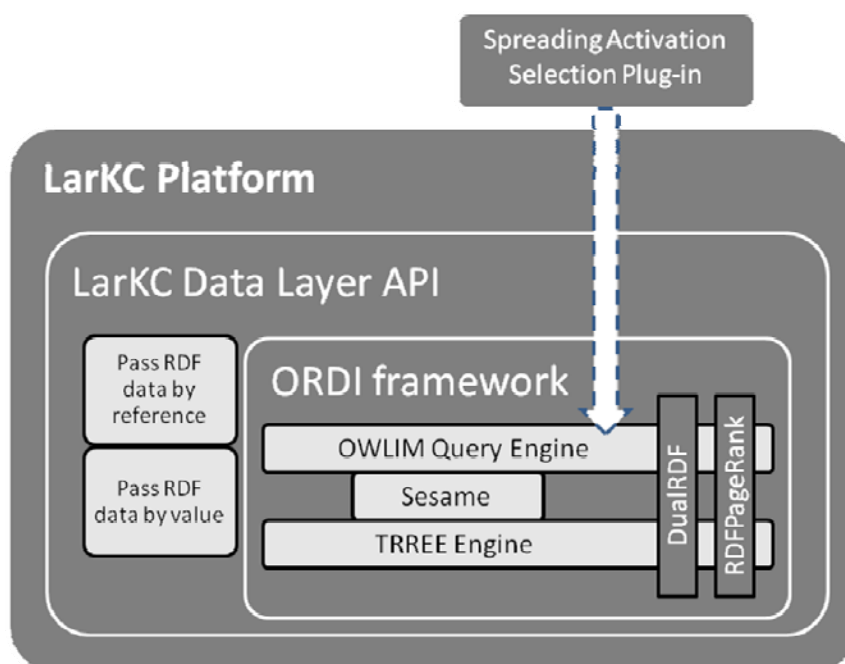


Figure 2. LarKC Data Layer Positioning and Architecture

1.3 Linked Data

“Linked data” is defined by Tim Berners-Lee, [3], as RDF graphs, published so that they can be navigated across servers by following the links in the graph in a manner similar to the way the HTML web is navigated. We provide a short introduction here to prepare the ground for understanding of the datasets used for validation of the data layer of the platform (see section 3).

The publishers of linked data should comply with four simple design principles:

1. Use URIs as names for things.
2. Use HTTP URIs, so that people can look up those names.
3. Provide useful information, when someone looks up a URI,
4. Include links to other URIs, so that they can discover more things.

In fact, most of the RDF datasets fulfil principles 1, 2, and 4 by design. Thus, the novelty of the design principles relates to the requirement for enabling Semantic Web browsers to load HTTP descriptions of RDF resources based on their URIs. To this end, data publishers should make sure that:

- the “physical” addresses of the published data are the same as the “logical” addresses, used as RDF identifiers (URIs);
- upon receiving a HTTP request, the server should return an RDF-molecule, i.e. a set of triples that describe the resource.

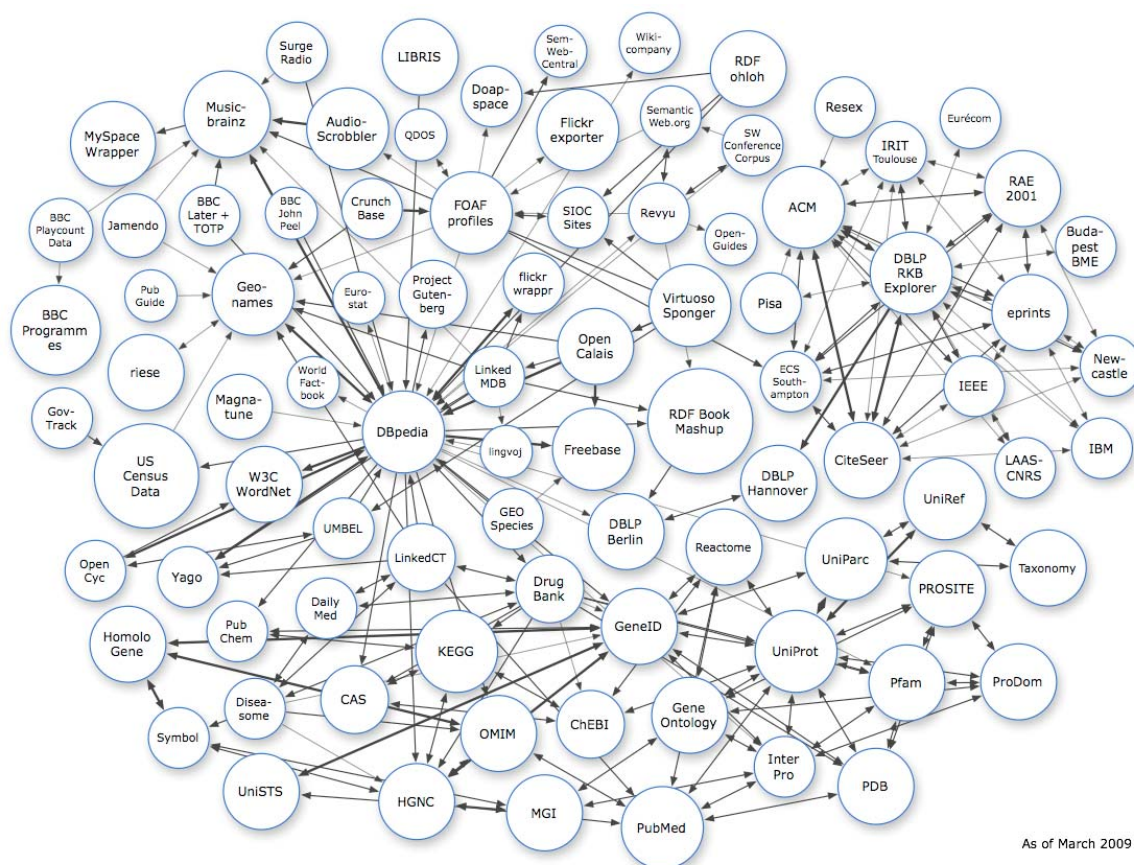


Figure 3. Map of the Datasets in Linking Open Data (LOD) Project
(from <http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData>)

Linked Open Data (LOD¹) is a W3C SWEO community project that aims at making data available to everyone. The goal of the project is to extend the web by publishing open data sets as RDF and by creating RDF links between data items from different data sources. The central dataset of the LOD is DBpedia – an RDF extract of the Wikipedia. Because of the many mappings between other LOD datasets and DBpedia, the latter serves as a sort of a hub in the LOD graph providing a certain level of connectivity. Among the linked LOD datasets are the following: the RDF representation of Wordnet (the most popular lexical knowledge base), Geonames (a database of all geographic features on Earth), World Factbook² (an RDF version of CIA’s resource), UniProt³ (the largest integrated database with protein and gene-related information), and OpenCyc⁴ (the most popular upper-level and general ontology).

UMBEL project deserves a special attention because it provides a consistent taxonomy interlinking DBpedia entities to OpenCyc concepts. In this way, the data from DBpedia (or

¹ <http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData>

² <http://www4.wiwiw.fu-berlin.de/factbook/>

³ <http://www.uniprot.org/>

⁴ <http://www.opencyc.org/>



from any other datasets connected to it) can be interpreted with respect to the semantics of OpenCyc.

Currently LOD contains more than 40 datasets, listed at page <http://esw.w3.org/topic/TaskForces/CommunityProjects/LinkingOpenData/DataSets>, with total volume above 4.7 billion statements, interlinked with additional 5 million statements. In section 3 we discuss in more details DBpedia, Geonames, UMBEL, and Wordnet datasets.



2 Data Layer Validation Framework

In the LarKC platform the data layer provides services typical for the so-called semantic repositories – RDF databases that may or may not provide lightweight inference support. Adequate benchmarking of semantic repositories is a complicated exercise, which requires proper conceptualization and structuring. In the remainder of this section we will try to provide such a conceptual framework, as further development of [26].

A wide range of links to resources related to benchmarking RDF repositories can be found at the page on “RDF Store Benchmarking” in the ESW wiki, maintained by W3C, at <http://esw.w3.org/topic/RdfStoreBenchmarking>.

2.1 Tasks

The major tasks and activities towards which the performance of semantic repositories needs to be benchmarked are:

- **Data loading**, including parsing, persistence, and indexing of both instance data and ontologies;
- **Query evaluation**, including query preparation and optimization and fetching;
- **Data modification**, which may involve changes to the ontologies and the schemata.

Inference is not a first-level activity in a semantic repository and in LarKC’s data layer. Depending on the implementation, it can affect the performance of the other activities. In the current implementation of the data layer, inference is performed during loading and affects its performance.

Modifications to the data and/or schemata (e.g. updating and deleting values or changing class definitions) represent another important class of the tasks performed against a semantic repository. Most of the contemporary RDF-related benchmark suites do not cover modification tasks, because their specifics, complexity, and importance can change considerably across applications and usage patterns.

2.2 Performance Factors

The performance of data **loading depends on** several factors:

- *Materialization* – whether and to what extent forward-chaining is performed at load time; complexity of the forward-chaining;
- *Data model complexity* – support for extended RDF data models, e.g. such including support for named graphs, is computationally more “expensive” as compared to the simple triple model;
- *Indexing specifics* – repositories may or may not create a variety of different indices in dependence of the datasets loaded, the foreseen usage patterns, hardware constraints, etc.;
- *Data access and location* – where the data is imported from, for instance, from local files, loaded from the network, etc.



There are several factors affecting the time and memory space, or more generally, the computing resources, required for **query evaluation**:

- *Deduction* – whether and to what extent backward-chaining is involved, whether it is recursive, etc.;
- *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- and disjunction-related clauses), the usage of operators that are tough to support through indexing (e.g. LIKE);
- *Number of clients* – number of simultaneous client requests;
- *Quality of results* – what is the quality of the results required in modalities where incomplete answers are requested.

Transaction size and level of isolation may also have serious impact on the performance of both loading and query evaluation. Although many semantic repositories provide some sort of transaction isolation, it is usually less comprehensive than the corresponding mechanisms in the mature RDBMS. Furthermore, transaction management in large scale system is usually carefully designed and tuned for each specific setup. Thus, we believe this aspect should not be included as a factor for investigation here.

2.3 Performance Dimensions

There are several parameters affecting the speed of a semantic repository:

- *Scale* – the size of the repository in terms of number of RDF triples (more generally, facts or atomic assertions).
- *Schema and data complexity* – the complexity of the ontology/logical language, the specific ontology (or schema), 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; sparse versus dense datasets; presence and size of literals; number of predicates used; usage of `owl:sameAs`, and other alignment primitives;
- *Hardware and software setup* – the performance can vary considerably depending on the version and configuration of the compiler or the virtual machine, the operating system, the configuration of the engine itself, and the hardware configuration, of course.

We should make a special note here regarding the different ways to measure the size of dataset loaded in a semantic repository:

- *Number of inserted statements (NIS)*: how many statements have been inserted in the repository;
- *Number of stored statements (NSS)*: how many statements have been stored and indexed by the repository. For engines using forward-chaining and materialization, the volume of the data to be indexed includes the inferred triples. For instance, the RDF/OWL representation of Wordnet expands after materialization from 1.9 million



(NIS) statements to 7.1 million (NSS). In the opposite direction, NSS can be smaller than NIS, when one and the same statement is inserted multiple times;

- *Number of retrievable statements (NRS)*: how many different statements can be retrieved from the repository. This number can be different from NSS when the repository supports some sort of backward-chaining. For instance, BigOWLIM performs `owl:sameAs` optimization, as discussed in section 3.5, which reduces considerably the NSS for linked data datasets.

2.4 Full-Cycle Benchmarking

We call *full-cycle benchmarking* a methodology that provides a complete picture about the performance of a repository with respect to the full “life cycle” of the data within the engine. At the high-level this means publication of data for both loading and query evaluation performance in the framework of a single experiment or benchmark run. In other words, full-cycle benchmarking requires load performance data (e.g. “5 billion triples of LUBM were loaded in 30 hours”) to be matched with query evaluation data (e.g. “... and the evaluation of the 14 queries took 1 hour on warm database.”).

The full-cycle benchmark methodology is important because loading and query performance are interdependent. More comprehensive indexing and forward-chaining take time during loading and respectively make the loading performance worse in order to facilitate faster query processing. In the extreme case, if query evaluation performance is not to be considered, loading of RDF could be as fast as the file-system could be in storing the input files locally with no overheads to maintain indices.

While there are differences across the different semantic repositories, a full-cycle run on LUBM, [21], or similar benchmark usually involves the following activities:

1. Loading input RDF files from the storage system
2. Parsing the RDF files
3. Indexing and storing the triples
4. Forward-chaining and materialization (optional)
5. Query parsing
6. Query optimization
 - a. Query re-writing (optional)
7. Query evaluation, involving
 - a. Backward-chaining (optional)
 - b. Fetching of the results

To provide correct answers to queries, a semantic repository should consider the semantics of the data. For instance, one cannot deliver correct results to the queries of LUBM without some form of reasoning. There are plenty of choices regarding the specific inference approach; inference can take place either during loading (step 4 above) or during query evaluation (steps 6a and/or 7a above).



Full-cycle benchmarking provides complete picture about the performance of the engine and its utility in real-world applications, showing the implications of the different design choices and implementation approaches.



3 LDSR as Test Dataset for Scalable Inference

Linked Data Semantic Repository (LDSR) was initially developed for the sake of testing of the components for selection and ranking in RDF graphs presented in [40]. LDSR was developed to meet the following requirements:

1. To be large enough to allow acquiring of meaningful feedback on the scalability of the implementations. It needed to be larger than 100 million statements, because most of the engines can manage smaller datasets in the main memory of a contemporary commodity server;
2. To present real-world data exposing “natural” connectivity patterns;
3. To be “predictable” and intuitive, i.e. to allow an average person to make assumptions, based on common knowledge and common sense, regarding the results of queries, ranking, and selection components;
4. To have some semantics attached to the data, or at least, to be easy to attach semantics to the data, so that further experiments with reasoning can be performed;
5. To integrate data from different sources, so that the graph structure is diverse and heterogeneous.

The use cases of the project already provide comprehensive datasets that allow for validation of the technology and evaluation of its exploitation potential. Still, these datasets do not match several of the above requirements and therefore were not appropriate for intermediate evaluation of early versions of selection and ranking components.

The natural choice of a dataset, matching the above requirements, is the collection of the Linking Open Data (LOD) datasets; it represents a set of interconnected datasets, published in accordance with the “linked data principles” (section 1.3). But the full collection of the LOD datasets was not suitable for the task, because some of the datasets were inappropriate for our purpose. For instance, due to the natural limitations of the accuracy of the extraction techniques used for its generation, the YAGO fragment of DBPedia contains plenty of faulty classifications of Wikipedia articles. Such inaccuracies are relatively small in number and probably are not a serious problem for humans exploring DBPedia. However, they can lead to significant inconsistencies for any type of reasoning. While reasoning with inconsistency is an interesting subject in principle, in such dataset inconsistent data would bring in distortion and complexity that is not desirable for many of its possible usages. We defined LDSR as a reason-able view to the LOD data, namely, a selection of the LOD dataset that meets the above stated requirements. A similar reason-able view, called Pathway and Interaction Knowledge base (PIKB), is defined and used in WP7a, [2]. While PIKB presents a very interesting experiment on its own, it is overly specialized and does not meet the predictability requirement.

This section presents LDSR together with statistics on its loading and discussion on specific problems related to the inference on top of it. LDSR is available for exploration and querying (through web UI form and SPARQL endpoint) at <http://www.ontotext.com/lcsr/>. Dumps of LDSR are also available upon request, but one should consider that it represents a collection



of the corresponding datasets from LOD. Therefore, acquiring up to date versions from the original owners of the datasets would be more appropriate.

3.1 Reason-able View to the Web of Linked Data

Linked Data Semantic Repository (LDSR) represents a reason-able view to the web data. In LDSR we selected several of the central datasets of the LOD project and loaded them in OWLIM. Reasoning was performed to "materialize" the facts that could be inferred from this data.

The overall setup is called Linked Data Semantic Repository (LDSR) to stress that: (1) Linked Data datasets are loaded together and (2) their semantics is used for inference. LDSR is a unique experiment with LOD data. Although many experiments were performed using LOD data, LDSR is pushing the frontiers in at least two directions:

- there are no published results of inference over such dataset of a size above 100 million statements;
- there are no published results of loading and inference over more than two datasets in a single repository of a size above 10 million statements.

Below we discuss the LOD datasets loaded in LDSR.

- **DBPedia**⁵ is an RDF dataset derived from Wikipedia. It is designed and developed to serve two purposes: (i) to provide as full as possible coverage of the factual knowledge that can be extracted with high precision from Wikipedia and (ii) to serve as a hub for the LOD project. Currently, DBPedia version 3.3 consists of about 360 million explicit statements.
- **GeoNames**⁶ is a geographic database that covers most of the significant geographical data categorized into feature classes such as state, country, province, populated place of any size, mountain, river, bridge, facility (e.g. oil field), etc. Each feature has a designator (type of the feature), geographic coordinates, and relations to other features (e.g. "parent" feature in which the feature is nested). At present, GeoNames database provides information on 6.5 million unique geographic features.
- **UMBEL** (*Upper Mapping and Binding Exchange Layer*) is a lightweight ontology structure that aims at organizing the web content and data⁷. The subject concepts and relationships in UMBEL are derived from OpenCyc. The extracted well-formed class hierarchy consists of about 20,000 classes, ranging from general philosophical notions like `TangibleThing` to very specific classes like `Abacloth`.
- **WordNet**⁸ is a lexical knowledge base that covers about 150,000 English words. WordNet defines the meanings of English words by grouping nouns, verbs, adjectives, and adverbs into sets of synonyms – the so called synsets. Each synset

⁵ <http://dbpedia.org/>

⁶ <http://www.geonames.org/>

⁷ <http://www.umbel.org/>

⁸ <http://wordnet.princeton.edu/>



expresses a distinct concept. The words linked to a given synset are synonyms with respect to the meaning expressed by this synset, which is a sort of lexical concept. A word can have multiple meanings, i.e. it can be associated with multiple synsets. The synsets are interlinked by means of semantic and lexical relations (links). The more general terms are associated with less general terms through hyponym-hypernym relations. In LDSR we use W3C's Wordnet RDF/OWL representation⁹.

- **CIA World Factbook**¹⁰ represents a collection of structured data, including statistical, geographic, political, and other information about all countries of the world;
- **Lingvoj**¹¹ is a dataset presenting multi-lingual descriptions of the most popular human languages; currently it contains information about more than 500 languages.

The connectivity in LDSR is assured by DBpedia, which provides links to GeoNames and Wordnet, and by UMBEL, which is linked to DBpedia.

We use ontologies to define the structure and the semantics of the datasets, loaded in LDSR. The following third-party ontologies and schemata are referred to or imported by these ontologies:

- **Dublin Core**¹² (DC) is a relatively small but very popular metadata schema. It defines 15 attributes (e.g. author/contributor, date of publication, language, etc.) that can be used for information resource description;
- **SKOS**¹³ (Simple Knowledge Organization System) represents a relatively simple RDF schema that allows description of taxonomies of concepts, linked to each other by any sort of subsumption hierarchy. The most important properties defined by SKOS are `skos:broader` and `skos:narrower`, defined as inverse of each other. The subsumption semantics of these relationships is more appropriate for encoding of "topic ontologies" and subjects classifiers as compared to the semantics of `rdfs:subClassOf`. The set-theoretic semantics of sub-class is not applicable for topic taxonomies. For example, while `AfricanLions` is a sub-topic of `Africa`, there is not a valid sub-class relationship between them.
- **RSS**¹⁴ is an RDF schema designed to enable syndication of machine-readable information about updates from web sites;
- **FOAF**¹⁵ is a project aiming to create a network of machine-readable personal profiles published on the web. In essence, the FOAF ontology defines attributes of these personal profiles, which allow publication of contact information and links to other profiles.

⁹ <http://www.w3.org/2006/03/wn/wn20/>

¹⁰ <http://www4.wiwiwiss.fu-berlin.de/factbook/>

¹¹ <http://www4.wiwiwiss.fu-berlin.de/factbook/>

¹² <http://purl.org/dc>

¹³ www.w3.org/2004/02/skos/

¹⁴ <http://web.resource.org/rss/1.0/spec>

¹⁵ <http://www.foaf-project.org/>



DC, SKOS, FOAF, and RSS ontologies are also loaded in LDSR because they are the basis for the semantic descriptions of the datasets. SKOS is not imported in the publicly available version of LDSR due to the problems with the category hierarchy of DBpedia, discussed in section 3.4.

3.2 Loading and Inference Statistics

The tests were performed using the owl-max ruleset of OWLIM reasoning on top of the LDSR dataset through forward-chaining. This ruleset delivers a combination of RDFS with incomplete OWL Lite, following the approach of Herman ter Horst, [12], to support the semantics of the OWL primitives in a tractable logical fragment. More information about this fragment can be found in section 4.2.1 of [11]. The rules and axioms in the owl-max ruleset are superset of those of language profile OWL-Lepton-I, defined in [11]. Based on our observation of the datasets and the definitions of the related ontologies, we believe that in this way we have performed complete inference against LDSR.

Table 1. LDSR Datasets

Dataset	Explicit Indexed Triples ('000)	Inferred Indexed Triples ('000)	Total # of Indexed Triples ('000)	Entities ('000 of nodes in the graph)	Inferred closure ratio
Schemata (SKOS, FOAF, RSS, DC) and ontologies (dbpedia, geonames)	10	7	17	5	0.7
DBpedia (SKOS categories)	2,233	262,734	264,968	952	117.6
DBpedia (owl:sameAs)	2,053	4,006	6,059	4,005	2.0
UMBEL	3,197	41,228	44,425	1,388	12.9
Lingvoj	20	112	132	18	5.7
CIA Factbook	161	40	202	53	0.2
Wordnet	1,943	5,236	7,179	842	2.7
Geonames	72,749	471,220	543,969	33,382	6.5
DBpedia 3.3 core	357,450	360,172	717,621	85,998	1.0
Total	439,815	1,144,755	1,584,571	126,642	2.6

3.3 Loading and Initial Inference Against LDSR

The statistics from loading and materialization of the implicit facts is presented in Table 1. The first column lists the datasets (or parts of them) in the order in which they were loaded into the repository. The number of triples listed in column “Explicit indexed triples” refers to the increased number of statements in BigTRREE indices after the dataset was loaded. Note



that some data providers claim that their datasets contain an amount of statements, slightly different from the one presented in the table.

We can summarize the results of the loading of LDSR as follows:

- Total number of inserted statements (NIS): 440 million;
- Total number of stored statements (NSS), including the implicit ones: 1,585 million;
- Total number of retrievable statements (NRS): 2,318 million.

The larger number of retrievable statements is a result of the `owl:sameAs` optimization, discussed in section 3.5; the optimization has “compressed” 734 million statements, reducing the size of the indices by 32%. Each explicit triple caused, on average, the materialization and indexing of 2.6 new implicit triples. There are 5.3 triples “retrievable” against a single explicit statement asserted.

The total time required for the loading and inference is 34 hours. Full-text indexing was performed during loading. The test is performed using a server with the following specifications: 2 x Xeon 5420 CPU (2.5 GHz), 64GB of RAM, OpenSolaris, JDK 1.6, RAID array of 8 SAS drives in RAID 5.

It is not a trivial task to load and perform inference over a large dataset like LDSR. We performed many experiments until we found the weird parts of the datasets and resolved all bottlenecks in the architecture. Below we list some problems discovered in the datasets:

- YAGO module of DBpedia includes a questionable class hierarchy and classifications. For instance, it is declared that Pleven (a large Bulgarian city, <http://dbpedia.org/page/Pleven>) is related through `rdf:type` to <http://dbpedia.org/class/yago/Village108672738>. The latter has the label “hamlet” and irrelevant relations to other nodes. After the forward-chaining such “strange” statements generate faulty statements. We decided not to include YAGO in LDSR.
- DBpedia is loaded on several “tranches” for the sake of speed optimisation.

3.4 DBPedia Category Hierarchy Refinements

A small subset of the SKOS vocabulary has been used in DBPedia to represent the categories hierarchy. Most importantly this includes the `skos:broader` relationship that defines a relation from one concept to another, broader than it.

There are about 478,378 categories in DBPedia ver. 3.3. Of those, 473,626 are explicitly declared and 4,752 undeclared, but mentioned as broader-to declared ones. DBPedia declares 896,803 broader relations among those categories.

While on one hand, having such category system as a result of collaborative community process is useful and interesting in several ways, it also causes some problems when it comes to the formal qualities of the hierarchy. Namely, it contains cycles and errors due to lexical ambiguity and other (human) factors. This prevents automated interpretation of the semantics of the categories hierarchy.



There have been previous attempts to remove the loops from the SKOS-based graph (e.g. the Cleaned Wikipedia Category Class Hierarchy – CWCC, <http://www.mail-archive.com/dbpedia-discussion%40lists.sourceforge.net/msg00216.html>) but they have been currently discontinued and the software developed does not seem to be publicly available.

Apart from the cycles, another problem with the hierarchy is that there are multiple subsumption relations that seem correct but turn up to be too vague to deliver useful results after materialization of the transitive closure of the `skos:broader` relationship. The shortcomings of these vague relationships are that they result in enormous number of super-categories, some of which are marginally useful. For instance, most of the subjects have a broader concept that is linked “upwards” with the general category for Philosophy, but in everyday life, having Melons defined as narrower category of Society comes a bit too abstract.

The task for cleaning up the category hierarchy, in order to make it “reason-able” or “materialise-able”, requires serious effort and dedication. As a start, one should come up with clear criteria about “good hierarchy” of Wikipedia categories, which is likely to have something to do with its intended usage, definition of metrics and test cases, etc. Here we present first results of several experiments with this category hierarchy aimed at acquiring an improved version within limited time and efforts.

Experiment 1: Discard broader cycles automatically

We devised an algorithm that first detects all simple cycles in the categories graph and starts discarding a random edge from them until all cycles are discarded (by discarding an edge we mean changing its predicate from `skos:broader` to `skos:related`). The algorithm tries not to remove more edges than needed.

During this experiment 2,165 simple cycles were detected, 1,321 of which were trivial – a category being marked as broader to itself (these were instantly discarded). The remaining 844 non-trivial ones were discarded by turning a random *broader* statement into a *related* statement.

Experiment 2: Manually discard mis-assigned broader statements causing cycles

At this step we took the 844 non-trivial cycles detected at the previous step and manually changed 868 relations from `skos:broader` statements to `skos:related`. This fixed all the cycles (it appears that in many cases a single erroneous broader statement causes more than one cycle in the graph). The resulting graph contained `skos:broader` paths of lengths ranging from 1 to 177.

Experiment 3: Remove system categories

The categories graph in the previous step contained quite lengthy paths which made it impossible to materialize in reasonable time. This is why we decided to remove all grouping categories such as “Categories by topic”. We applied a very simple heuristic in order to determine which categories could be considered system categories – exactly those that contain “ by ” in their name. We did not remove those categories. Instead, we changed the links to them (i.e. from narrower categories) to *related* instead of *broader* and, in the cases



when the system categories had a non-system parent category, we redirected the previously existing *broader* links to the parent categories. This operation increased the number of broader statements to 1,099,364 but reduced the length of the paths to maximum of 171. All the remaining cycles (which increased to 4,433) were discarded by the manually prepared rules.

Experiment 4: Solve parental ambiguities

Even the reduced graph resulting from the previous step was hard to materialise. This is why we proceeded to solve categories parental ambiguities (i.e. cases when a category has more than one `skos:broader` parents). On a first iteration we chose to remove all ambiguities when they existed. This reduced the graph too much and left us with as much as 105,202 broader relations. We considered this approach too limiting and decided to solve some of the ambiguities by choosing the ambiguous parent that had the most article instances in common with the child category. This disambiguation strategy left us with as much as 417,042 broader relations. This dataset was now both materialise-able and quite sensible. On a further iteration we increased the number of ambiguous parents chosen to the two most appropriate ones (i.e. having most instances in common or having more instances in general if the first criterion resulted in too much categories). This increased the number of broader relations to 728,882.

The resulting dataset contains 728,882 `skos:broader` and 582,087 `skos:related` statements. The inferred closure of the `skos:broader` relations contain around 262 million SKOS-related statements (`broader`, `narrower`, `related`, `broaderTransitive`, `narrowerTransitive`, and `semanticRelation`).

3.5 owl:sameAs Optimisations

The loading of LDSR benefited greatly from a specific feature of the BigTRREE engine that allows the engine to handle efficiently `owl:sameAs` statements. `owl:sameAs` is a predicate declaring that two different URIs denote one and the same resource. Most often, it is used to align the different identifiers of the same real-world entity used in different data-sources. For instance, the URI of Vienna in DBpedia is <http://dbpedia.org/page/Vienna>, while the URI in Geonames is <http://sws.geonames.org/2761369/>. DBpedia contains the following statement

```
(S1) dbpedia:Vienna owl:sameAs geonames:2761369
```

It declares that the two URIs are equivalent. `owl:sameAs` is probably the most important OWL predicate when it comes to merging data from different data-sources.

Following the formal definition of OWL, whenever two URIs are declared equivalent, all statements that involve one of the URI should be “replicated” with the other URI. For instance, in Geonames the city of Vienna is defined as part of <http://www.geonames.org/2761367/> (the first-order administrative division in Austria with the same name), which in its turn is part of Austria (<http://www.geonames.org/2782113>):

```
(S2) geonames:2761369 gno:parentFeature geonames:2761367
```



Deliverable 5.5.2

(S3) `geonames:2761367 gno:parentFeature geonames:2782113`

As long as `gno:parentFeature` is a transitive relationship, in the course of the initial inference, it will be derived that the city of Vienna is also part of Austria:

(S4) `geonames:2761369 gno:parentFeature geonames:2782113`

Due to the semantics of `owl:sameAs`, from (S1) it should be inferred that statements (S2) and (S4) also hold for Vienna, when it is referred with its DBpedia URI:

(S5) `dbpedia:Vienna gno:parentFeature geonames:2761367`

(S6) `dbpedia:Vienna gno:parentFeature geonames:2782113`

These are true statements and when querying RDF data, no matter which one of the equivalent URIs is used in the explicit statements, the same results will be obtained. When we consider that Austria also has equivalent URI in DBpedia, we should also infer that:

(S7) `dbpedia:Vienna gno:parentFeature dbpedia:Austria`

(S8) `dbpedia:Austria gno:parentFeature dbpedia:Austria`

As a result, for a pair of equivalent URIs, a number of new statements will be derived. Probably, there are many concepts with more than two URIs. For instance, Vienna has URI also in UMBEL, which is also declared equivalent to the one in DBpedia.

`owl:sameAs` is transitive, reflexive, and symmetric, so a set of N equivalent URIs N^2 `owl:sameAs` statements will be generated for each pair of URIs. Thus, although `owl:sameAs` is useful for interlinking RDF datasets, its semantics causes considerable inflation of the number of implicit facts that should be considered during inference and query evaluation (either through forward- or through backward-chaining).

To overcome this problem, BigTRREE handles `owl:sameAs` in a specific manner. In TRREE indices, each set of equivalent URIs (equivalence class with respect to `owl:sameAs`) is represented by a single super-node. In this way, BigTRREE does not inflate the indices while it retains the possibility to enumerate all statements that should be inferred using the equivalence. Special care is taken to ensure that this “trick” does not hinder the ability to distinguish explicit from implicit statements.

This optimisation allows BigTRREE to handle efficiently large datasets where `owl:sameAs` is extensively used. In the case of LDSR, this technique allows dealing with more than 2.1 billion statements at the computational costs required for 1.5 billion statements.



4 State of the Art in Semantic Repositories

In this section we present the most relevant publicly available benchmark results about several of the outstanding semantic repositories. Before analyzing evaluation data, it should be noted that there are very few full-cycle benchmarking (see section 2.4) results reported for datasets larger than one billion statements. Most often there is data about loading and indexing performance, which are not matched by query evaluation data. In many cases this seems to be driven by practical problems with the benchmark environments. For instance, in query evaluation for large datasets in the LUBM, [21], benchmark is impractical, as some of the queries return millions of results, producing a load pattern that is not quite likely to appear in real-world applications. Either due to inappropriate benchmark design or for other reasons, full-cycle benchmarking for very large scale experiments requires a lot of time and resources as such experiments often take several days for a single run. In other cases, there are interesting datasets, the loading and inference for which represent a challenge. However, those are not matched with well-thought and standardized set of queries that could be used for benchmarking. For the above listed reasons, we include here some loading performance results that are not matched with query evaluation data. We do this only in cases when there are sufficient details given about the experiment, providing evidence that loading performance is not optimized in a manner that will make query evaluation infeasible or impractical.

Here we will not include results on scalable DL reasoning, because we are not aware of significant developments, beyond the state of the art reported in section 4.1 of, [26].

The section starts with an introduction of the datasets and the benchmarks used, the engines included here, and the benchmarking environments (hardware, OS, etc.) Next we continue with overview and analysis of the current state of the art of loading and query evaluation performance of semantic repositories.

4.1 Benchmarks and Datasets

There are several benchmarks and datasets used nowadays as measuring sticks for evaluation of the performance of semantic repositories. Few of the most popular ones are presented briefly in this section.

4.1.1 Lehigh University Benchmark (LUBM)

The most popular benchmark for semantic repositories with support for RDF and OWL is Lehigh University Benchmark (LUBM), defined in [21]. 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 DLP¹⁶. The LUBM benchmark defines 14 queries that are used to check the query evaluation correctness and 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, distributed in 1,000 RDF/XML files with total size of 600 megabytes. For the

¹⁶ Please refer to [8] and [15] for further information on these and other OWL dialects.



purposes of scalability measurements many groups have used the LUBM generator to create bigger datasets – there are multiple results of such experiments in the following sections.

4.1.2 UniProt

UniProt¹⁷ (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, [37], is an RDF representation of the database with respect to OWL ontology, expressed in a sub-language of OWL Lite, that is more expressive than OWL Horst, but still tractable. It represents one of the largest datasets distributed in RDF and OWL. Processing UniProt is often used as benchmark for scalability and reasoning capabilities of semantic repositories.

While in the summer of 2007 the size of the RDF representation of UniProt was about 384M statements, today it includes above 1.5 billion unique explicit statements. The RDF graph defined in the UniProt ontology is highly interconnected, which has significant impact on the loading and reasoning speed of semantic repositories. Today UniProt is one of the central datasets in the bio-medical part of the Linking Open Data initiative (see section 1.3).

The RDF representation of UniProt was used for benchmarking of Jena and AllegroGraph (two of the engines in our study). It is also part of the Pathway and Interaction Knowledge Base (PIKB), [2], developed in WP7a and used in both of the life science use cases of LarkC.

4.1.3 DBPedia and Other LOD Datasets

As discussed in section 1.3, DBPedia is a dataset representing in RDF the Infobox of Wikipedia together with other information related to or derived from Wikipedia. It serves as a connectivity hub of the Linked Open Data (LOD) initiative, as there are multiple relationships (mostly through `owl:sameAs`) between it and other datasets. The diversity of the information represented in DBPedia and the fact that it represents encyclopaedic knowledge, makes it an excellent resource for benchmarking of semantic repositories.

Within LarkC we use it for benchmarking as part of the Linked Data Semantic Repository, which includes also several other datasets from LOD, as presented in section 3. Based on our experiments, DBPedia ver. 3.3 includes 362 million unique statements, excluding the `owl:sameAs` links. The total number of statements indexed in LDSR after materialization is 1,584 million.

4.1.4 Berlin SPARQL Benchmark (BSBM)

Berlin SPARQL Benchmark, [4], evaluates the performance of query engines in an e-commerce use case: searching products and navigating through related information. Randomized “query mixes” (of 25 queries each) are evaluated continuously through client application that communicates with the repository through SPARQL endpoint. The benchmark allows evaluation with respect to changing sizes of the dataset and different number of simultaneous users.

¹⁷ <http://www.uniprot.org/>



Although created for benchmarking of SPARQL engines, the design of BSBM favours the relational databases and other raw-store-based implementations, as long as it deals with a single fixed data schema and uniform dense data representation. Generally, RDF databases are designed to deal efficiently with diverse data, integrated from multiple data sources, encoded against different data schema, resulting in sparse data tables in relational databases. BSBM does not require any inference. Still, the benchmark is useful for evaluating the SPARQL support of the engines and their ability to support multiple simultaneous clients.

4.2 Engines

After the introduction of the Sesame framework, several of the outstanding RDF engines are briefly presented in alphabetical order in the subsequent sections.

4.2.1 Sesame

Before starting the overview of the various engines, it is important to mention that several of the engines rely on the Sesame RDF database framework, developed by Aduna B.V. under the OpenRDF¹⁸ open-source project. Sesame is ranked by multiple independent evaluations (e.g. [36]) as the most efficient RDF repository framework. Semantic repositories like OWLIM and BigData (see the subsections below) use Sesame as a library, taking advantage of its APIs for storage and querying, as well as the support for a wide variety of query languages (e.g. SPARQL and SerQL) and RDF syntaxes (e.g. RDF/XML, N3, Turtle). Other engines like Virtuoso and AllegroGraph use it just for the sake of interoperability.

4.2.2 4store

4store¹⁹ is an open-source RDF storage engine designed to run in cluster setup of up to 32 nodes. It is implemented in ANSI C99 and is currently available for UNIX-like systems only.

The RDF repositories created with 4store are managed through a set of command line utilities that allow importing of RDF data in various formats, querying using SPARQL, backup, and other maintenance tasks. The 4store distribution also includes a stand-alone SPARQL HTTP protocol server as a standard interface to the 4store repository.

4store is most popular as the backend of Garlik (<http://www.garlik.com/>) – a company dealing with online identity and personal information protection in UK. 4store is advertised with the fact that at Garlik it is “holding and running queries over databases of 15GT, supporting a Web application used by thousands of people.” There are several other interesting performance and scalability claims made about 4store²⁰, which would have been

¹⁸ <http://www.openrdf.org/>

¹⁹ <http://www.4store.org/>

²⁰ Some of the results are reported at <http://www.4store.org/>, while others were announced directly at <http://esw.w3.org/topic/LargeTripleStores>, in both cases with very little or no information about the benchmark setup and organisation.



a nice complement to the summary prepared in section 4.4, if there was sufficient information provided about them.

4.2.3 AllegroGraph

AllegroGraph RDFStore²¹ is an RDF database with support for “SPARQL, RDFS++, and Prolog reasoning from Java applications”. In addition to the “basic” DBMS functionality, it offers specific support for geo-spatial data handling and social network analysis.

AllegroGraph does not perform reasoning and materialization during loading. The so-called RDFS++ reasoning²² can be switched on during query processing. The semantics supported in this way is comparable to OWL Horst (see [18]). RDFS++ reasoning allows AllegroGraph to deliver correct results on LUBM benchmark. Version 3.2 of AllegroGraph supports the so-called “RDFS++ dynamic materialization”²³, which, based on the very laconic explanations of the vendor, requires building some extra indices on the fly, whenever those are required for query evaluation. This approach helps AllegroGraph report excellent results²⁴ on query evaluation in LUBM(8000). However, given the limited information available, it is very hard to judge how generic this technique is and for what queries, what semantics, and what datasets it can be expected to work efficiently.

One of the major developments in AllegroGraph 3.0 is the so-called “federation” that allows grouping multiple stores (running locally or on a remote machine) within a single virtual store. Federation has the potential to considerably speed up the loading process, as the work is effectively distributed across multiple stores. As reported in [19] and [1], loading LUBM(8000) on a 4-CPU machine with a virtual store combining four local stores is three times faster than performing the same load in a single, unified (i.e. non-federated) store. On the other hand [1] reports that query evaluation gets 2-3 times slower in the federated store. In the loading performance analysis in section 4.4 we have included the results from loading LUBM(8000) and UniProt (a rather old version of it) from both the single-instance and the federated runs. The single-instance results are important because they need to be considered as a match of the excellent query evaluation results achieved by AllegroGraph on LUBM(8000).

Further [19] reports that “*On Amazon's EC2, AllegroGraph loaded and indexed 10 Billion Triples derived from 1 Billion Telecom CDRs (Call Detail Records) into 10 large instances, 4 parallel loads per instance, in 6.19 hours*”. While this is a very impressive result on its own, we will leave it without comment and will not include it in the scalable reasoning table and diagram below. The above quotation is the only information publicly available and there are no indications about the complexity of the dataset or any data about queries that were evaluated on top of this set.

²¹ <http://agraph.franz.com/allegrograph/>

²² <http://agraph.franz.com/support/documentation/3.0/reasoner-tutorial.html>

²³ <http://www.franz.com/agraph/allegrograph/dynamic-materialization.lhtml>

²⁴ http://www.franz.com/agraph/allegrograph/agraph_bench_lubm.lhtml



4.2.4 BigData

<http://www.bigdata.com/blog/BigData>²⁵ is an open-source distributed B+Tree database, designed to accommodate large RDF repositories and scale horizontally on commodity hardware. Scaling is achieved by index partitioning across the nodes of the cluster. As the data evolves, partitions of the indices might get split, moved, or joined transparently and asynchronously to the client. A centralized metadata index takes care of locating the index partitions in the cluster. As discussed in [38], BigData is designed to support various modalities of transactional isolation through multi-version concurrency control (MVCC, [35]) and aims to support load balancing internally.

BigData integrates with Sesame 2.0 platform (see sub-section 4.2.1) to achieve SPARQL support and therefore relies on it for query parsing but does query optimization based on the fast key-range counts supported natively by the B+Tree architecture.

BigData supports RDFS+ inferencing backed by a hybrid approach of materializing the entailments of some rules (forward-chaining) at load time and using backward-chaining for others at query time. Other features of BigData include statement-level provenance and full-text indexing.

The latest evaluation results²⁶ about BigData represent scale-out experiments in a cluster environment of 15 blade servers, each of them with the following parameters: Xeon 3GHz, Quad Core, 4GB of RAM, 32-bit environment. Thus, this is a system with 60 cores and 60GB of RAM in total. The expected cost of this configuration is at least three times higher than the limit of \$10,000, which we set as a constraint, allowing measurement of similar environments. Still, we include this scale out data in the tables and charts below with a note about the hardware used. Latest available benchmarks of BigData report that the loading of 12.7B triples took approximately 12 hours.

4.2.5 BigOWLIM

OWLIM is a semantic repository implemented in Java and packaged as a Storage and Inference Layer (SAIL) for the Sesame RDF database (see sub-section 4.2.1). OWLIM is based on TRREE – a native RDF rule-entailment engine. The standard inference strategy is forward-chaining and materialization. The supported semantics can be configured through rule-set definition and selection. The most expressive pre-defined rule-set combines unconstrained RDFS and OWL Lite (see [18]). Custom rule-sets allow tuning for optimal performance and expressiveness.

The two major varieties of OWLIM are SwiftOWLIM and BigOWLIM. In SwiftOWLIM, reasoning and query evaluation are performed in-memory, while, at the same time, a reliable persistence strategy assures data preservation, consistency, and integrity. BigOWLIM is the “enterprise” variety that deals with data and indices directly from disc or other file storage, which allows it to scale much better. Further, BigOWLIM’s indices are

²⁵ <http://www.systap.com/bigdata.htm>

²⁶ Some of the results are reported at the product blog (<http://www.bigdata.com/blog/>), while others were only briefly announced at <http://esw.w3.org/topic/LargeTripleStores>.



specially designed to allow efficient query evaluation against huge volumes of data. SwiftOWLIM can manage millions of explicit statements on desktop hardware. Given an entry-level server, BigOWLIM can handle billions of statements and serve multiple simultaneous user sessions.

BigOWLIM 3.1 is the current release of the engine and delivers the following improvements:

- Based on BigTRREE 3.1, supporting named graphs and triple sets;
- It is compliant with SPARQL, through Sesame 2.0;
- Its persistence and indexing strategy allow for smoother and more efficient scalability in the range of billions of triples.

Results of benchmarking of various versions of OWLIM can be found in the OWLIM Benchmarking section²⁷ of the Ontotext web site. The results can be summarized as follows:

- BigOWLIM can perform reasoning against 12 billion explicit statements; it loads a dataset of LUBM(90,000) at average speed of 11,5 KSt/sec;
- BigOWLIM can deal with 20 billion statements on a single server worth less than \$10,000. This is the total number of the statements inserted in the indices after materialization of LUBM(90k). Although direct loading of the same number of statements without reasoning would be faster, we use the data from this run as scale-up statistics about BigOWLIM;
- BigOWLIM can deal with 1 billion statements on a desktop machine worth \$2,000; it takes it less than 5 hours to load the dataset; loading with inference and materialization takes 14 hours;
- The full-cycle run of LUBM(8000), including loading, inference, and query evaluation, takes only 15.2 hours.

Apart from the scale-up experiments, the most interesting results for BigOWLIM are related to the LDSR dataset, presented in section 3. In contrast to the LUBM scale-up experiments, LDSR is way more challenging for several reasons:

- It includes fairly diverse real-world knowledge;
- There are more than 100 thousand different predicates used;
- There are plenty of literals, of various size;
- There are long chains of transitive properties, most notably: the SKOS categories from DBPedia (see section 3.4); the nesting of the geographic features from GeoNames; the hypernyms from Wordnet; the class hierarchy from UMBEL;
- There is a large number of `owl:sameAs` statements.

All the above specifics result in loading and inference complexity, which is considerably higher than these for LUBM.

²⁷ <http://www.ontotext.com/owlim/benchmarking/index.html>



4.2.6 DAML DB

DAML DB is an older name of the engine of BBN Technologies, which currently appears as part of Parliament²⁸ – an open-source knowledge base management system that implements a high-performance storage engine, compatible with the RDF and OWL standards. It is usually combined with query processing frameworks, such as Sesame (sub-section 4.2.1) or Jena (sub-section 4.2.7), to implement a complete data management solution with support for SPARQL query language.

Although there are no recent evaluation results published, DAML DB still seems to be one of the very few systems that can demonstrate a full-cycle benchmark results (see section 2.4) on test like LUBM(8000). Unfortunately, its results, provided in [36], lack concrete measurement data; there are 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 our diagrams below, we made imprecise approximations for the figures behind the diagrams. We would be happy to update this document with more concrete data, shall such be made available.

4.2.7 Jena TDB

TDB²⁹ is an open-source RDF storage layer for the Jena semantic web Java framework³⁰. It is implemented purely in Java and can be accessed through Jena APIs or through several separately provided command line scripts. Paired with Jena, TDB provides a single-machine, non-transactional RDF storage and query environment that can be accessed over SPARQL protocol when running inside the Jena-based Joseki HTTP server. SPARQL queries over TDB are made possible through Jena's ARQ SPARQL query engine.

A TDB repository can be operated by 32-bit and 64-bit JVM without any format migration being necessary (direct consequence of Java's architecture independent types). However, in 64-bit mode TDB uses memory-mapped files to access its repository binary representation (data and indices). This is reported to contribute to a much better performance compared to the 32-bit case where data caching is handled by the TDB engine itself. Another benefit from relying on the operating system to do the file caching is the dynamically determined amount of memory used by the engine for disk caches.

TDB is equipped with different SPARQL query optimizers (e.g. fixed and statistical) that aim to optimize SPARQL queries on per-repository basis. The fixed optimizer only considers the number of variables in query's triple pattern in its reordering decision. The statistical optimizer relies on rules that approximate the number of results to be expected from a triple pattern (those rules could be either written manually or generated by the engine). TDB interprets RDF simple types (e.g. `xsd:integer`), which makes it possible to optimize SPARQL range filters.

²⁸ <http://www.bbn.com/technology/knowledge/parliament>

²⁹ <http://jena.hpl.hp.com/wiki/TDB>

³⁰ <http://jena.sourceforge.net/>



Latest benchmarks³¹ available for Jena TBD report that 1.7B triples were loaded in 36 hours on an unspecified 64-bit machine.

4.2.8 ORACLE

ORACLE offers RDF support as part of the Spatial option of its DBMS since version 10g R2. As presented in [42], in version 11g this support is improved in several ways, including:

- Support for the so-called OWL Prime dialect, which is comparable with the owl-max semantics of OWLIM, [31]. The Pellet DL reasoner is integrated for T-Box (schema level) inference.
- The efficiency of RDF loading and inference is considerably improved.

ORACLE supports RDF models with named graphs, i.e. it can be seen as a quadruple store. The semantics to be used for inference is defined in terms of the so-called rule-bases, which essentially represent sets of entailment rules. Inference is implemented in terms forward-chaining and materialization – the results are stored in the so-called rule-indices. The initiative regarding inference is given to the user, who should take care to explicitly:

- Force inference, i.e. generation of the rule indices;
- Invalidate the rule indices when necessary, i.e. after statements are deleted.

The latest results from benchmarking ORACLE 11g are published in [42]. One should consider summing up the times for the different steps. For instance, the times reported for LUBM(8000) are as follows:

- Bulk-load: 30 h. 43 min;
- Loading into staging table: 11 h. 32 min. (when proper correctness checks are performed by the RDF parser);
- Inference (OWL Prime): 11 hours. Multi-threaded inference runs 3.3 times faster as compared to single-threaded one on a quad-core CPU;

We shall acknowledge two important circumstances:

- The LUBM(8000) results for ORACLE are measured on desktop computer; while most the other results are measured on servers. In fact, the configuration used for inference is comparable to the workstation used for benchmarking BigOWLIM on LUBM(8000);
- ORACLE is also the most “economic” with respect to the RAM usage – the above results reflect the inference run with 4GB of RAM, but results measured on 2GB systems, suggest graceful degradation of the performance when less memory is available;
- The results reported for loading and for inference come from different machines; for some reason there are no public results for loading times at the same CPU where ORACLE achieves its best inference time.

³¹ <http://seaborne.blogspot.com/2008/06/tdb-loading-uniprot.html>



4.2.9 Virtuoso

OpenLink Virtuoso³² is a “universal server” offering diverse data and metadata management facilities, e.g. XML management, RDBMS integration, full-text indexing, etc. The core engine of Virtuoso is a relational database engine with numerous RDF oriented adaptations in data types, index layout and query optimization.

Virtuoso does not have built-in materialization of entailment during loading. Instead, it supports the semantics related to sub-classes, sub-properties, and `owl:sameAs` at run time, through backward-chaining on the basis of a specified RDF schema. This way, the user is not required to rewrite queries. Materialization is possible by writing SPARQL statements to generate implied triples.

Several aspects related to the development and the configurations of efficient distributed RDF repositories are investigated by the developers of Virtuoso in [14]. This works provides a lot of interesting experimental data, which should be invaluable reading for everyone aiming and scalable RDF managements.

The developers of Virtuoso report in [13] the results of various “distributions of labour” between materialization and query expansion. As expected, it appears that extensive materialization can save a lot of computations during querying, while the same query completeness is achieved. In principle, this type of inference can be implemented on top of any DBMS – the complexity of the inference depends on the complexity of the query language supported. The principle concern here is that the semantics is not enforced by the engine, but it is rather a matter of hand crafted queries that may or may not correctly reflect the semantics of the ontology language primitives.

The data provided in [13] leaves several questions open:

- The queries of LUBM are modified, so, it is not clear: (i) how the query evaluation performance compares with this or other engines benchmarked with the original LUBM queries and (ii) whether the query results are truly complete and whether the implemented inference mechanisms are correct and sufficient for a full LUBM run.
- No performance data is provided for entailment and materialization, i.e. there are no indications about the performance and efficiency of the various inference configurations that were put on test.
- Implementing the semantics of transitive properties via query-based entailment and materialization, as presented in [13], requires recursive query evaluation. However, there are no comments on the implementation of such behaviour in Virtuoso.

Version 6.0 is the latest generation of the Virtuoso engine, which supports distributed RDF database management. The results³³ from the loading of LUBM(8000) in a cluster of 8 instances running on a single dual-CPU, eight-core, server represent the fastest load on this benchmark on a server equipment worth less than \$10,000.

³² <http://www.openlinksw.com/virtuoso/>

³³ <http://www.openlinksw.com/dataspace/vdb/weblog/vdb%27s%20BLOG%20%5B136%5D/1563>



4.2.10 YARS2

Results related to indexing and querying 7 billion statements in cluster of 16 machines are published for YARS2, [22]. 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 experiments with YARS2 are not discussed below.



4.3 Benchmark Configurations and Environments

Table 2 presents a summary of the tools and the benchmark environments included in the loading and query performance overviews in the subsequent sections.

Table 2. *Systems and Configurations*

Tool	Version	Configuration
AllegroGraph	AllegroGraph 64-bit RDFStore, ver. 3.2	Server 2xOpteron 844 (1.8 GHz) CPUs, 16GB RAM, 2xSATA Server 4xOpteron 844 (1.8 GHz) CPUs, 16GB RAM, 2xSATA Server 2xXeon 5410 (2.33GHz,qc), 64GB RAM, storage: no info.
OWLIM	BigOWLIM 3.1	Server 2xXeon 5420 (2.5GHz,qc) CPUs, 64GB RAM, 8xSAS (onap) Desktop Core i7 (2.93GHz,qc) CPU, 12GB RAM, 3xSATA (osol)
Virtuoso	Open-source version 5.5 Virtuoso 6 Cluster	Server 2xXeon 5335 (2GHz) CPUs, 16GB RAM, 6xSATA drives Server 2xXeon 5410 (2.33GHz,qc) CPUs, 16GB RAM, 6xHDD
ORACLE	ORACLE 11g	Desktop Core 2 Q6600 (2.4GHz) CPU, 8GB of RAM, 3xSATA drives
DAML DB	Ver. 2.2.1.2, via Sesame ver. 1.2.6	Server 2xXeon E5345 CPUs (2.33 GHz), 16GB RAM, 6 TB storage (probably RAID over 10+ drives)
BigData	0.8b	Configuration of 15 blade servers: " Each has 32G of RAM, 8 cores, and around 100G of local disk available to the application". Overall cost can be estimated above \$40 000

The equipment used for the runs of the different systems falls into four categories:

1. Desktop hardware with 8-12 GB of RAM. For ORACLE and BigOWLIM, such hardware is sufficient for management of datasets in the range of one billion statements, e.g. LUBM(8000) and LDSR (see sections 4.1.1 and 3.3).
2. Dual-CPU (8-core) servers with 16 GB of RAM, like the ones used in the benchmarks of DAML DB, Virtuoso, and some of the runs of AllegroGraph. While these configurations can offer much better performance, compared to the desktops, when handling queries from multiple clients, their performance for loading LUBM(8000) is quite comparable to the one of the desktops.
3. Dual-CPU (8-core) servers with 64 GB of RAM, like the ones used in some of the benchmarks of AllegroGraph and BigOWLIM. These machines seem to represent the optimal hardware setup for semantic repositories that could be purchased for less than \$10 000 at present. The quad-Opteron configuration used in the benchmarks of federated AllegroGraph, do not seem to have an advantage against the above mentioned dual-CPU configurations.
4. Cluster configuration with cost above \$20 000, like the one used in the BigData benchmarks. Most likely, the environment referred in the 4Store scalability claims (section 4.2.2).



In general, the reader should consider the performance metrics below as indicative figures providing evidence about the current state of the art in scalable reasoning, rather than as comparative analysis between the tools. One should take into account, that configurations that appear comparable, based on the short descriptions published by most of the vendors, can demonstrate 2-3-fold differences in performance on the basis of different storage, operating and file systems, and configuration parameters (e.g. sector size).

4.4 Loading Performance

Let us 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 materialized statements);
- Inference, namely materialization through forward-chaining, for OWLIM, DAML DB, and ORACLE.

Table 3 presents the loading time and speed of the engines for specific runs and datasets. For each run we 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 rate**: average of the Forward-Chaining and the Parsing and Indexing Complexity indices. It represents a combined measure for the complexity of the loading task;
- **Forward-chaining semantics**: what ontology language or fragment is used for forward-chaining and materialization during the loading, if any;
- **Forward-chaining complexity-index**: our subjective estimate about the complexity of the reasoning during this run;
- **Parsing & Indexing Complexity**: our 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.

The “-mat” suffix in the results of BigOWLIM marked as LUBM(40k)-mat and LUBM(67k)-mat indicates that the scale on this rows denotes the total number of indexed statements, including the materialized ones. So, this is just a different way to present the results of the LUBM(40k) and LUBM(67k) loads. Note that loading time is the same. We made this exception for the following reasons:

- the extraordinary scale of these results; LUBM(67k)-mat represents the most scalable RDF loading experiment published so far;
- the speed indicated in this way is lower than the speed at which an engine would simply load 20 billion explicit statements without inference;
- Simply loading LUBM without inference is not very useful in BigOWLIM, because without materialized results we will not be able to evaluate the query performance.

**Table 3. Loading Performance**

Semantic Repository	Dataset / Run	Scale (mill. ex.st.)	Speed (KSt./sec.)	Overall Compl exity Rate	Loading Time (hours)	Forward-Chaining Semantics	Materiali zation Complex ity Rate	Parsing & Indexing Complexit y
BigOWLIM 3.1	LUBM(1k), osol	138	25,617	15	1.50	OWL-Horst	20	10
BigOWLIM 3.1	LUBM(1k), osol	138	76,159	5	0.50	none	0	10
BigOWLIM 3.1	UNIPROT	1,150	15,212	20	21.00	OWL-Horst	30	10
BigOWLIM 3.1	PIKB	1,466	9,694	28	42.00	OWL-Horst	45	10
BigOWLIM 3.1	LDSR1	357	14,167	28	7.00	OWL-Horst	40	10
BigOWLIM 3.1	LDSR2	440	3,608	40	33.88	OWL-max	60	20
BigOWLIM 3.1	LUBM(8k), osol	1,068	42,381	9	7.00	RDFS	10	7
BigOWLIM 3.1	LUBM(8k), osol	1,068	20,631	15	14.38	OWL-Horst	20	10
BigOWLIM 3.1	LUBM(8k), osol	1,068	66,196	5	4.48	none	0	10
BigOWLIM 3.0	LUBM(20k)	2,760	44,516	5	17.22	none	0	10
BigOWLIM 3.0	LUBM(50k)	6,675	43,914	5	42.22	none	0	10
BigOWLIM 3.1	LUBM(16k)	2,136	13,267	15	44.73	OWL-Horst	20	10
BigOWLIM 3.1	LUBM(24k)	3,204	13,039	15	68.26	OWL-Horst	20	10
BigOWLIM 3.1	LUBM(32k)	4,272	12,772	15	92.92	OWL-Horst	20	10
BigOWLIM 3.1	LUBM(40k)	5,343	12,493	15	118.80	OWL-Horst	20	10
BigOWLIM 3.1	LUBM(67k)	8,949	12,015	15	206.90	OWL-Horst	20	10
BigOWLIM 3.1	LUBM(40k)-mat	9,083	21,238	10	118.80	OWL-Horst	0	20
BigOWLIM 3.1	LUBM(67k)-mat	15,218	20,432	10	206.90	OWL-Horst	0	20
AllegroGraph 3.2	UNIPROT	234	19,403	5	3.35	none	0	10
AllegroGraph 3.2 Fed.	UNIPROT	234	55,085	5	1.18	none	0	10
AllegroGraph 3.2	LUBM(8000,0)	1,107	39,423	5	7.80	none	0	10
AllegroGraph 3.2 Fed.	LUBM(8000,0)	1,107	13,667	5	22.50	none	0	10
Virtuoso, ver. 6.0 cluster	LUBM(8000,0)	1,068	110,559	5	2.68	none	0	10
ORACLE 11g	LUBM(1k,0)	138	7,986	5	4.80	none	0	10
ORACLE 11g	LUBM(8k,0)	1,068	7,047	5	42.10	none	0	10
ORACLE 11g	LUBM(8k,0)	1,068	5,484	15	54.10	OWL-Prime	20	10
ORACLE 11g	LUBM(25k,0)	3,300	5,343	15	171.56	OWL-Prime	20	10
DAML DB	LUBM(8k)	850	10,266	14	23.00	OWL-Horst	20	7
DAML DB	LUBM(8k)	220	6,111	14	10.00	OWL-Horst	20	7
Jena TDB	UNIPROT v.13.3	1,516	11,698	5	36.00	none	0	10
BigData - cluster	LUBM(8k)	1,155	51,041	5	6.28	none	0	10
BigData - cluster	LUBM(8k)	1,155	32,201	10	9.96	RDFS	10	10
BigData - cluster	LUBM(?)	5,000	138,889	5	10.00	none	0	10
BigData - cluster	LUBM(?)	9,000	80,645	5	31.00	none	0	10
BigData - cluster	LUBM(?)	10,400	61,466	5	47.00	none	0	10



LDSR1 corresponds to an earlier version of the dataset, close to the one presented in [40]. LDSR2 represents the load of the version of the dataset documented in section 3.2. The latter load is more challenging than the former for the following reasons:

- FOAF and SKOS schemata were loaded, which triggers considerable inference complexity;
- The load was made with OWLIM’s ruleset owl-max, instead of OWL Horst;
- A newer and larger version of DBPedia is loaded;
- Full-text indexing is switched on during the load – this is the reason to indicate loading and indexing complexity 20 for this load in Table 3).

The so-called “scalable inference map” presented below, 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. For better visibility, we have split the map into two figures: Figure 4, showing loads up to 2 billion statements and Figure 5 showing all loading results.

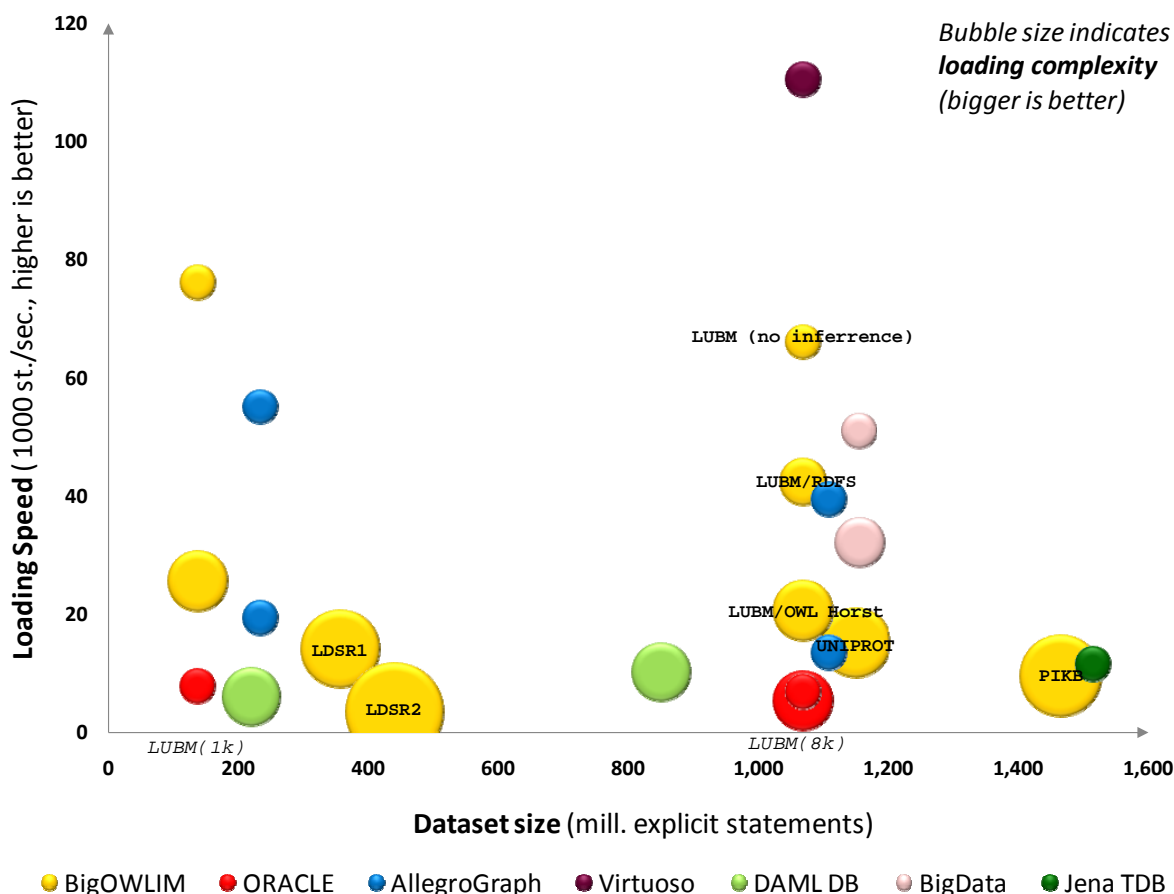


Figure 4. Scalable Inference Map – up to 2MSt.

One can see that the results for loading are comparable, taking into account that the engines differ in features. Taking BigOWLIM’s results for LUBM(8000), one can observe how the difference in the supported semantics can alter the loading time almost by a factor of three.

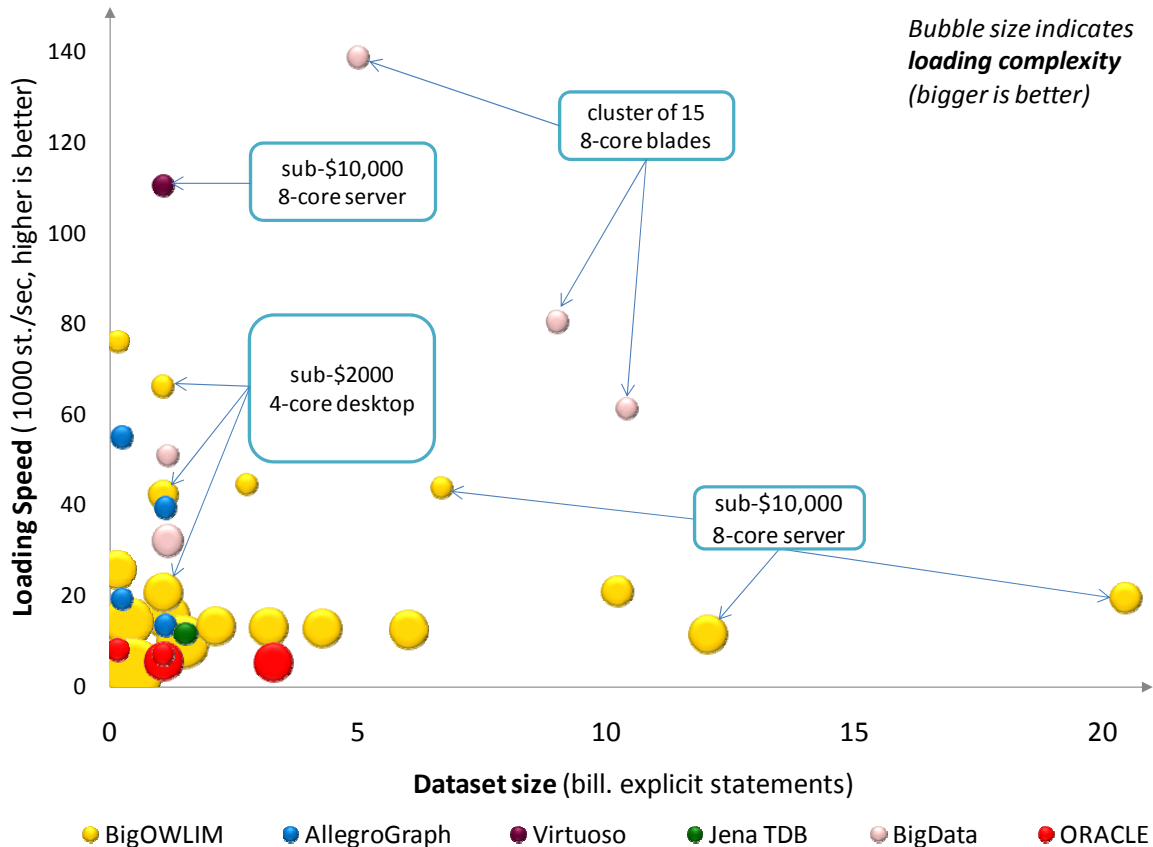


Figure 5. Scalable Inference Map – The Big Picture

4.5 Query Evaluation

Below we provided publicly available results about query evaluation performance of the engines presented in section 4.2 in the framework of two benchmarks: BSBM (see section 4.1.4) and LUBM (see section 4.1.1). Benchmarking query evaluation against a real-world datasets like those in LDSR (section 3) is preferable over to the synthetic datasets. However, there is still no well-thought and evaluated set of standard queries to the LOD datasets that can be used for such purposes.

4.5.1 BSBM Results

Recent results from evaluation of few of the most popular engines are published in [5]. Figure 6 provides a comparison between results from our evaluation of BigOWLIM 3.1 and results for other systems from [5] regarding query evaluation with growing number of simultaneous clients (1, 4, 8) against 25-million statement version of the BSBM dataset.

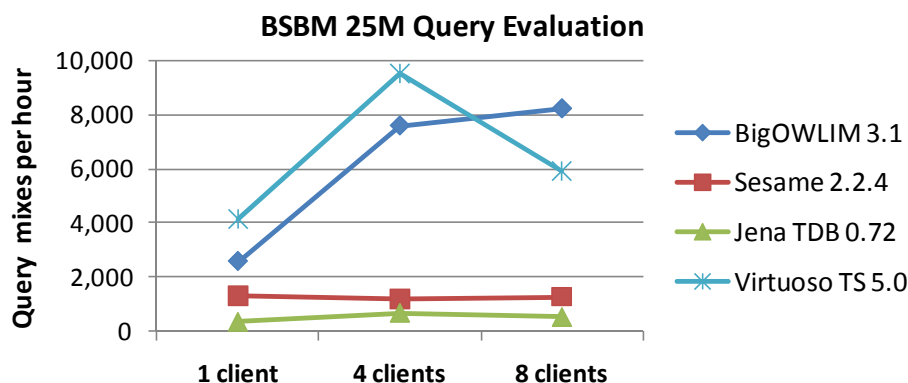


Figure 6. BSBM Query Results

Given larger number of simultaneous clients, BigOWLIM and Virtuoso can deliver considerably larger overall throughput, almost proportional to the number of CPU cores of the system: BigOWLIM results are acquired on a desktop (osol) with a quad-core CPU with hyper-threading support; Virtuoso’s results are acquired on a desktop quad-core CPU without hyper-threading.

One should consider that the 25M dataset of BSBM fits into the main memory of the benchmark configurations used, thus effectively, all queries are evaluated in-memory. Unfortunately, there are no publicly available results on multi-client evaluation for larger versions of BSBM.

Table 4. Query Performance LUBM(8000), 1 billion statements

Query No	# of results	BigOWLIM 3.1 (onap)		BigOWLIM 3.1 (osol)		AllegroGraph 3.2	
		Time (msec)	QTPR (msec)	Time (msec)	QTPR (msec)	Time (msec)	QTPR (msec)
1	4	23	5.8	52	13.0	4	1.0
2	2,528	251,992	99.7	873,197	345.4	54,964	21.7
3	6	26	4.3	25	4.2	2	0.3
4	34	8	0.2	28	0.8	14	0.4
5	719	38	0.1	29	0.0	37	0.1
6	83,557,706	84,333	0.0	174,777	0.0	104,794	0.0
7	67	38	0.6	71	1.1	9	0.1
8	7,790	113	0.0	132	0.0	178	0.0
9	2,178,420	523,215	0.2	1,460,862	0.7	117,303	0.1
10	4	27	6.8	24	6.0	5	1.3
11	224	4	0.0	23	0.1	8	0.0
12	15	6	0.4	33	2.2	14	0.9
13	37,118	1,662	0.0	121,980	3.3	47	0.0
14	63,400,587	63,998	0.0	157,568	0.0	27,990	0.0
Average		66,106	8.4	199,200	26.9	21,812	1.9



4.5.2 LUBM Results

Although there is plenty of data about loading performance and scalability of LUBM (see section 4.4), in very few cases query evaluation has been benchmarked within the same environment that was used for dataset loading.

In Table 4 we present the results of query evaluation of the LUBM benchmark with 8000 universities. Two sets of results are provided for BigOWLIM 3.1 – for server (onap) and desktop (osol). The results of AllegroGraph 3.2, [20], are acquired on a server comparable to the one used for BigOWLIM. The engines are described in section 4.2; specifications of the benchmark environments are available in section 4.3.

Considering that some of the LUBM queries return very large number of results for large datasets, in earlier versions of this analysis we have introduced the query-time-per-result (QTPR) metric, where the query evaluation time is normalized with respect to the size of the result set. Average QTPR represents a better overall score of the query performance on LUBM as the performance of queries with large result sets does not dominate the score as in the case of using average query evaluation time. QTPR appearing as 0.0 indicates a value below 0.1 milliseconds.

As expected, given a large dataset the query performance of the server equipment is considerably better than this of the desktop, which should most probably be attributed to the better storage system and data buses between the CPUs and the main memory.

The query performance reported for AllegroGraph looks very good, especially if one considers that, based on the vendor information (see section 4.2.3), “dynamic materialization” is performed during query evaluation. The major gain in terms of average QTPR and evaluation time comes from queries #2 and #9 – probably the two most challenging ones in the benchmark. We have tried to reproduce these results in internal benchmarks of AllegroGraph, but we did not manage to get close to the figures reported by the vendor. This could be a matter of suboptimal configuration of the engine or different benchmark setup.



5 Validation Targets and Future Work

Validation targets for the data layer of the LarKC platform should be aligned with the requirements of LarKC's use cases and its plug-ins. The requirements of the life-sciences use cases, at their current state of development, are already matched as the work in these workpackages is already exploiting the data layer for development and usage of the PIKB dataset (see section 1.1). Requirements related to streaming reasoning (WP6) are still a matter of further analysis and investigation.

LDSR was designed as a test dataset in work-package WP2, [40], where plug-ins that reduce the scale of the datasets are implemented. The results presented in section 3.2 represent an organic development of this dataset, which would make it more suitable as a testbed for other plug-ins as well as for entire inference pipelines. The most notable development is that the latest version of the LDSR requires (or provides ground for) much more comprehensive reasoning, compared to the one presented in [40]. This is because now all the ontologies and schemata, related to the dataset, are used (the SKOS and FOAF schemata were initially disregarded). Further, the experiments now use entailment against more expressive rule-sets. A simple and direct evidence about the fact that the latest version of LDSR is a more challenging reasoning target is that its deductive closure is 3 times bigger (1.5 billion statements versus 0.5 billion statements) and it represents a higher degree of "expansion" of the original dataset (now it is 5.3 times bigger than the original data, as compared to 1.7 times for the earlier version).

Finally, as the level of ambition in LarKC is to achieve reasoning performance and scalability that go far beyond the current state of the art, LarKC's data layer should also push the frontiers with respect to semantic repositories. The overview of these engines, presented in section 4, allows us to summarize the current state of the art as follows:

- There are already several engines that can deliver loading and indexing speed (without materialization) in the range of 100,000 statements/sec., against datasets of few billion statements (see section 4.4);
- The query evaluation performance against such datasets is in the range of few milliseconds per single query result (QTPR, see section 4.5.2);
- Several of the engines are already mature enough to be able to handle multiple-client loads with throughput that scales up in parallel with the multi-threading capabilities of the test setup (see section 4.5.1);
- Full materialization with respect to complex real-world data can be performed for datasets in the range of one billion statements (consider PIKB) with a speed in the range of 10 000 statement/sec;
- Dealing with datasets in the range of 20 billion statements and simple reasoning with more than 10 billion statements are already possible.

Although some of the results presented in section 4.4, involve cluster setups far beyond the entry level for database servers, those results were also matched by other engines on commodity hardware where the test setups cost less than \$10,000.



It is important to note that distributed semantic repositories are already available from several vendors. While such repositories deliver the best results with respect to loading performance, one should consider two observations:

- single-server parallelization still seems to be most cost-effective way of getting optimal load speed on commodity hardware – the top-performance loads of both AllegroGraph and Virtuoso use multiple instances of the engine running on a single server;
- the best query evaluation results are delivered by non-distributed setups.

5.1 Performance Targets

This section introduces measurable performance targets for semantic repositories, based on the discussion on the relevant tasks and factors (section 2), and on the analysis of the current state of these systems (section 4). Our goal is to provide measuring sticks, which could facilitate the development of this type of technology over the next couple of years.

Generally, the targets defined earlier in [26], still seem to be valid. We will not include targets related to incomplete reasoning in this version of the validation, because the latter should be evaluated at the level of the whole LarkC platform, not at the level of the data layer. The design of measures for the performance of the platform as a whole will be discussed in D1.4.1 “Initial framework for measuring and evaluating heuristic problem solving”, which is due M18 of the project.

We define an updated set of 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 1 msec;

SRPT2: Complete UniProt reasoning. The UniProt dataset should be loaded. Queries comparable in complexity to the set published at <http://www.linkedlifedata.com>, should be evaluated; the queries should involve full-text search constraints.

Target: loading speed 40 000 explicit st./sec.; average QTPR below 100 msec.

SRPT3: Complete linked data reasoning. Dataset similar to LDSR with size above 5 billion statements should be loaded. The queries to be evaluated should involve: data from different data-sources; non-trivial reasoning (recursive query evaluation or rule-entailment); full-text search constraints.

Target: loading speed 20 000 explicit st./sec.; average QTPR below 100 msec.

For all the targets above, eligible results should match the requirements for full-cycle benchmarking set in section 2.4. In summary, both loading and query evaluation results should be achieved in the course of a single benchmark, i.e. the indices produced during loading should be used in the same environment and configuration for query evaluation.

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

- The targets shall be met on a commodity hardware setup 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 newer version of UniProt and LDSR, which differ in size or complexity from the version available in the summer of 2009;
- Ordering and relevance ranking is irrelevant to the above targets.

The scalability targets defined here are just several 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.

5.2 Data Layer Distribution Considerations

The LarKC Data Layer is a central architecture component that serves the persistence and the communication of large sets of RDF data to the platform plug-ins. The overall LarKC platform scalability and performance parameters are subjected to the data storage parameters presented in section 4.4. By design, its restriction to a single computer system will limit the overall platform scalability. To better understand the challenge of overcoming this problem, we focus on the use cases, where two common problem patterns are distinguished:

- Volumes of information - the scenarios of WP7a and WP7b use cases utilize a massive knowledge base to answer complex life sciences and health care related questions. The number of client requests is expected to be relatively small and limited to domain experts only.
- Volumes of simultaneous user requests - the use case WP6 “Urban Computing” is expected to have extreme number of simultaneous user requests, where only a limited dataset is relevant to the user’s question.

The data distribution is a non-trivial task that requires significant compromises in the engineering design. The CAP theorem, [6], summarizes the results of significant amount of theoretical and experimental work in distributed database systems by stating the trade-offs between consistency, availability, and partition-tolerance. To resolve the two dimensions of the problem, we consider two alternative distribution strategies driven by: partitioning and redundancy with load balancing.

Data Partitioning

In the database systems two general approaches to data partitioning come up from the established distributed database management systems research: horizontal and vertical partitioning. The horizontal data partitioning approach should be interpreted as partitioning a dataset across several repository nodes whereas no schema limitations apply to any of the nodes. In contrast to the horizontal dataset partitioning, a vertical partitioning approach would assign different parts of the data schema to different repository nodes. Further on, queries for any type of statement would be redirected to the respective repository node. A careful analysis of the repository schema (when available) might even extend this approach further and place statements that are “close” together (e.g. Geonames subset of LDSR) within a single node (when possible). Such clustering would allow for whole sub-queries to be executed within a single node. It would therefore avoid the transfer of intermediate



results between the repository nodes and the central query processing node only to complete the query.

Hybrid schemes are also possible for datasets that are both horizontally and vertically too large to fit in a single computational node. Such a partitioning scheme however would complicate considerably the query processing subsystem, which would have to take into account splits and joins in both dimensions while planning the query execution.

Redundancy and Load Balancing

Data replication is a traditional approach for boosting the read performance of a DBMS at the cost of redundancy and write propagation complexity. In a classic scenario, several slave nodes are assigned incoming read requests by a central master node that performs any sort of load balancing technique (e.g. round robin) to distribute the load evenly across the slaves. Writes are executed on the master node and propagated to the slaves in the background. Such a setup is very appropriate in situations when a lot of read requests occur while write requests are rare or clustered together in large batches (for example if a large dataset is initially loaded in the repository). In such situations the resource-intensive replication procedure will not be necessary most of the time, while a theoretically linear scalability will take place on the reading side.

As already mentioned, a direct consequence of the CAP theorem [6] is that data consistency might have to be sacrificed if availability and partitioning are required. In the context of the above mentioned LarkC's use-cases, such a trade-off is fairly acceptable. One might argue that data inconsistency could be a serious issue in certain types of real-world production-level applications. However reasonable this argument might be in general, we believe that an eventually-consistent and highly scalable distributed system could be very well suited for a data layer of an incomplete reasoning system (such as LarkC), where result completeness is not a requirement by design.

In addition to the various multi-node data distribution approaches, the above one should also consider parallelization at thread level within a single computer system. These days even commodity desktop hardware is equipped with dual or even quad-core CPUs that are better utilized by multi-threaded implementations whenever such are appropriate. One possible candidate for multi-threading is the “federated join” approach in which the outermost loop of the query evaluation routine (i.e. the one traversing the outermost triple pattern of the query) is split among several threads that operate over equal parts of the repository. This is another case that would be appropriate in scenarios when reads occur a lot and writes are rare.



6 Conclusion

We provided framework for validation of the data layer of the LarKC platform, based on analysis of the various tasks and aspects related to storage, querying, and lightweight inference. The major tasks are loading, inference, and query evaluation. The major performance factors are scale, reasoning complexity, number of simultaneous clients, software and hardware environment. Specific targets are set on the basis of analysis of the current state of the art in the field of scalable RDF databases and light-weight inference. To calibrate the validation framework and determine the targets we performed:

- Experiments with the current version of the data layer with respect to the couple of datasets related to the life sciences case studies (PIKB) and benchmarking of plug-ins (LDSR);
- Analysis of the current state of the art in scalable RDF databases and light-weight reasoning.

Both PIKB and LDSR can be seen as reason-able views to the Linking Open Data (LOD) cloud of interconnected public datasets. They represent a selection of several datasets and different parts of them, optimized so that they can be loaded in a single semantic repository and so that complete inference can be performed on top of them. PIKB represents a collection of about 20 popular biomedical datasets – more details about it can be found in the relevant deliverables in WP7a. LDSR includes few of the most central datasets in LOD, namely DBPedia, Geonames, Wordnet, and others. It has total size of about 440 million explicit statements, complemented with additional 1,145 million statements, after materialization of their inductive closure. Inference was performed with respect to OWLIM's owl-max ruleset, which represents an extension of OWL-Lepton-I defined in LarKC. LDSR emerged as a testbed for experimentation with retrieval, selection, and ranking components and plug-ins in WP2. It was further developed in WP5 to serve as a testbed for the overall project. Based on published results, LDSR seems to be the largest body of common sense knowledge used for inference.

Overall, the evaluation demonstrates that the implementation of LarKC platform's data layer is very well positioned with respect to the other outstanding engines in the highly competitive niche of the so-called semantic repositories. The scalability of TRREE matches the requirement of the use cases and practically all popular datasets. In order to reach the next level of scalability, the data layer of LarKC will have to provide efficient means for distributed data management. This requires advancing the frontiers of the state of the art in distributed RDF database management, where the best current results involve several machines and report scalability and performance results that are easy to match with a single machine at a fraction of the cost.



References

- [1] Aasman, J.: *AllegroGraph 4.0 - Industry's First Real Time RDF Store*. Presentation at Semantic Technologies Conference, San Jose, 2009.
- [2] Andersson, B., Momtchev, V.: *LarkC Requirements summary and data repository*. LarkC project deliverable D7a.1.1, 2008.
- [3] Berners-Lee, T.: *Design Issues: Linked Data*, 2006.
- [4] Bizer, Ch., Schultz, A.: *Benchmarking the Performance of Storage Systems that expose SPARQL Endpoints*. In: Proceedings of the 4th International Workshop on Scalable Semantic Web knowledge Base Systems, SSWS2008, 2008.
- [5] Bizer, Ch., Schultz, A.: *Berlin SPARQL Benchmark Results*. Document version 1.2 of 23 March, 2009.
- [6] Brewer, E.: *Keynote at Symposium on Principles of Distributed Computing*, 2000.
- [7] Brickley, D., Guha, R.V, eds.: *Resource Description Framework (RDF) Schemas*. W3C Recommendation, 2004.
- [8] Carroll, J., Bizer, Ch., Hayes, P., Stickler, P.: *Named graphs, Provenance and Trust*, 2005.
- [9] Cunningham, H., Roberts, A., Li, Y., Kiryakov, A., Schooler, L., Quesada, J., Neth, H., Della Valle, E., Braga, D., Zhong, N.: *Deliverable D2.1.1 Selection and Retrieval Methods*. LarkC project deliverable, 2008.
- [10] Dean, M; Schreiber, G. – editors; Bechhofer, S; van Harmelen, F; Hendler, J; Horrocks, I.; McGuinness, D. L; Patel-Schneider, P. F.; Stein, L. A. (2004). *OWL Web Ontology Language Reference*. W3C Recommendation, 10 Feb, 2004.
- [11] Della Valle, E., Dell'Aglio, D. Celino, I.: *1st periodic report on data and performances*. LarkC project deliverable D6.4., 2009.
- [12] DeWitt, D. J; Stonebraker, M. (2008). *MapReduce: A major step backwards*. *The Database Column*. January 17, 2008.
- [13] Erling, O. (OpenLink Software). *LUBM and Virtuoso*, 24 Jul, 2009.
- [14] Erling, O; Mikhailov, I.: *Towards Web-Scale RDF*. ISWC2008 submission as of 23 July, 2009.
- [15] Fensel D., van Harmelen, F., Andersson, B., Brennan, P., Cunningham, H., Valle, E. D., Fischer, F., Huang, Z., Kiryakov, A., Lee, T. K., School, L., Tresp, V., Wesner, S., Witbrock, M., Zhong, N.: *Towards larkc: a platform for web-scale reasoning*. In *Proceedings of the IEEE International Conference on Semantic Computing (ICSC 2008)*, Santa Clara, USA, 2008.
- [16] Fensel, D., van Harmelen, F.: *Unifying reasoning and search to web scale*. *IEEE Internet Computing* 11(2), 2007.
- [17] Fischer, F., Bishop, B.: *Empirical Analysis of the Initial Knowledge Representation*



- Formalism*. LarkKC project deliverable D1.1.2, 2009.
- [18] Fischer, F; Keller, U; Kiryakov, A; Huang, Z; Momtchev, V; Simperl, E.: *Initial Knowledge Representation Formalism*. LarkKC project deliverable D1.1.3, 2008.
- [19] Franz Inc. (2008). *High-performance Storage and Querying*, as of 24 Jul, 2009.
- [20] Franz Inc.: *AllegroGraph RDFStore Version 3.2 LUBM Benchmark Results*, as of 19 Jul, 2009.
- [21] Guo, Y; Pan, Z; and Heflin, J.: *An Evaluation of Knowledge Base Systems for Large OWL Datasets*. Journal of Web Semantics, 3(2), 2005, pp. 158-182, 2004.
- [22] Harth, A; Umbrich, J; Hogan, A; Decker, S.: *YARS2: A Federated Repository for Querying Graph Structured Data from the Web*. In Proc. ISWC, 2007.
- [23] Horrocks, I., Patel-Schneider, P. F., Bechhofer, S., Tsarkov, D.: *OWL Rules: A Proposal and Prototype Implementation*. Journal of Web Semantics 3 (2005), pp. 23-40, 2005.
- [24] Kerrigan, M., Bradesko, L., Fortuna B.: *Rapid Prototype of the LarkKC*. LarkKC project deliverable D5.2.1, 2009.
- [25] Kim, K., Celino, I., Della Valle, E., Dell'Aglio, D., Huang, Y., Hauptmann, W.: *Templates of periodic report on data and performances*. LarkKC project deliverable D6.2, 2009.
- [26] Kiryakov, A.: *Measurable Targets for Scalable Reasoning*. LarkKC project deliverable D5.5.1, formerly titled "Definition of validation goals for the prototyping phase", 2008.
- [27] Li, Y., Cunningham, H., Roberts, A., Kiryakov, A., Momtchev, V., Greenwood, M., Aswani, N., Damjanovic, D. *Selection Components (report accompanying two software deliverables)*. LarkKC project deliverable D2.2.1, 2.5.1, 2009.
- [28] MacCartney, B; McIlraith S. A; Amir, E; Uribe, T.: *Practical partition-based theorem proving for large knowledge bases*. In Proc. of IJCAI, 2003.
- [29] Manola F., Miller, E. (eds.), *RDF Primer. W3C Recommendation*, 10 Feb 2004.
- [30] Ontotext Lab. *OWLIM – Pragmatic OWL Semantic Repository. Presentation.*, as of 18 Jun, 2008.
- [31] Ontotext Lab. *RDF(S), Rules, and OWL Dialects*. Introductory web page.
- [32] Ontotext Lab: *SwiftOWLIM. System Documentation*. Version 2.9.1 from 30 Sep, 2007.
- [33] Oracle Corp.: *Oracle Semantic Technologies Inference Best Practices with RDFS/OWL*. An Oracle White Paper February, 2008.
- [34] Orud'hommeaux, E., Seaborne, A.: *SPARQL Query Language for RDF. W3C Recommendation*, 15 Jan, 2008.
- [35] Reed, D. P.: *Naming and Synchronization in a Decentralized Computer System*. MIT dissertation, 1978.
- [36] Ruhloff, K; Dean, M; Emmons, I; Ryder, D; Sumner, J. (2007). *An Evaluation of Triple-Store Technologies for Large Data Stores*. In Proc. of Scalable Semantic Systems Workshop SSSW, 2007.



- [37] Swiss Institute of Bioinformatics. *UniProt RDF*, 2009.
- [38] SYSTAP LLC: *Bigdata: Approaching web scale for the semantic web*, 2009.
- [39] ter Horst, H. J. *Combining RDF and Part of OWL with Rules: Semantics, Decidability, Complexity*. In Proc. of ISWC 2005, Galway, Ireland, LNCS 3729, pp. 668-684, November 6-10, 2005.
- [40] Todorova, P., Kiryakov, A., Ognyanoff, D., Peikov, I., Velkov, R., Tashev, Z.: *Spreading Activation Components*. LarkC project deliverable D2.4.1, 2009.
- [41] Weithöner, T., Liebig, T., Luther, M., Böhm, S. *What's Wrong with OWL Benchmarks?* SSWS, 2006.
- [42] Wu, A.; Lopez, X.: *Building Enterprise Applications With Oracle Database 11g Semantic Technologies*. Presentation at Semantic Technologies Conference, San Jose, 2009.