SLUBM: An Extended LUBM Benchmark for Stream Reasoning

Tu Ngoc Nguyen, Wolf Siberski

L3S Research Center
Leibniz Universitat Hannover
Appelstrasse 9a D-30167, Germany
{tunguyen, siberski}@l3s.de

Abstract. Stream reasoning is now emerging as a hot topic in the context of Semantic Web. As the number of data sources that continuously generates data streams emulating real-time events are increasing (and getting more diverse, i.e., from social networks to sensor networks), the task of exploiting the temporal aspects of these dynamic data becomes a real challenge. Stream reasoning is another form of the traditional reasoning, that works with streaming (dynamic, temporal) data over an underlying (static) ontology. There have been many existing reasoning systems (or newly proposed) trying to cope with the problems of stream reasoning but there is yet no standard to measure the performance and scalability of such systems. This paper proposes a benchmarking system, which is an extension to the well-known benchmark for traditional reasoning, Lehigh University Benchmark (LUBM), to make it work for stream-based experiments while retaining most of the LUBM's old standards.

1 Introduction

RDF Data have been prevailing in the era of Semantic Web and now has become a major part of the World Wide Web. Together with the growth in size of RDF data, there appears a constant need of processing heterogeneous and noisy RDF in a stream-based approach, as the problem is faced in many different applications [1,2,3] such as social networks, feeds, sensor data processing, network monitoring, network traffic engineering and financial markets. There are several approaches [4,5,2,6,3] in coping with the challenges of stream reasoning, where the robustness against inconsistent and continuous RDF data and the ability to preserve useful past computations and incremental reasoning are crucial. In this paper, we propose an enhancement of the existing predominant benchmark for static inference engines, LUBM [7] to make it a new version for assessing the performance of these engines in the context of stream reasoning. For simplicity, we will use the terms stream reasoning and incremental reasoning interchangeably throughout the paper, although they do not necessarily have the same meaning.

There are two main contributions of this paper. First, we introduce a benchmark that is specifically designed for testing the performance of incremental

reasoning engines. To the best of our knowledge, this is the first benchmark that provides a practical measuring method for assessing the capabilities of such systems. Secondly, we conduct extensive experiments using our benchmark to measure the performance of a number of state-of-the-art reasoning engines of different types (i.e., OWL-DL, rule-based) which are *streamable* or can be adapted to process stream data. We are confident this will aid the improvement of these tested engines toward the domain of stream reasoning.

2 Related Work

Stuckenschmidt *et al.* [6] analyzed the characteristics of existing reasoning systems towards expressive stream reasoning and proposed possible extensions to overcome their drawbacks. They pointed out the technologies of C-SPARQL, RETE and incremental reasoning in description logics (DL) can support incremental reasoning but are not explicitly meant for processing data streams with newer facts are more relevant. Accordingly, they proposed an ideal system concept to cope with the complexity of reasoning with rapid changes. The proposed architecture is used as the backbone of this paper, in building up our stream reasoning benchmark.

There are plenty of different works that provide a benchmarking assessment for RDF storage systems. We classify these systems into two different categories: *reasoning*-based and *SPARQL*-based, since the semantics of RDF and RDFS are omitted in the SPARQL specification [8].

2.1 Reasoning-based Benchmarks

LUBM [7] is a benchmark for OWL knowledge base systems, consisting of a *university* ontology and an ABox of arbitrarily large scalability. LUBM provides a university database where the components i.e., university, departments, professors, students can be polynomially generated. The benchmark ontology is expressed in OWL Lite language. However, the reasoning in response to the proposed queries does not require to support OWL Lite reasoning. Alternatively, it can simply perform by realizing the Abox [9]. LUBM offers 14 well-designed test queries[1] that fully cover the features of a traditional reasoning system. One disadvantage of LUBM is whereas the data volume can grow polynomially by the number of university generated, the complexity of these queries remains due to the sparseness of the data, reflecting by no connection links between the university instances. UOBM [10] is an extension version of LUBM and addresses the incompleteness of LUBM in fully supporting OWL Lite or OWL DL inference. UOBM provides two different versions of the university ontology in OWL Lite and OWL DL. External links between members in different university instances are also added to create connected graphs rather than the old isolated ones in LUBM. This exponentially raises up complexity for scalability testing.

[1] `http://swat.cse.lehigh.edu/projects/lubm/query.htm`

2.2 SPARQL-based Benchmarks

SP^{2}Bench [11], is a framework which measures the performance of triple stores in response to specified query language, SPARQL. The framework consists of two parts, the *data generator*, provide arbitrarily DBLP-like models in RDF format and the collection of 17 *benchmark queries*, designed to test all strengths and weaknesses of SPARQL engines. Berlin SPARQL benchmark (BSBM) [12], addresses the untapped part of SPARQL-to-SQL rewriting sytems of SP^{2}Bench. It provides a set of measurements of which, deep comparisons between native RDF stores and systems rewriting SPARQL-to-SQL against its relational databases are carried out. BSBM emulates e-commerce scenarios in which the query mix is designed to follow a customer's search and navigation pattern while looking for a product. On the other hand, OpenRuleBench [13] provides a suite of benchmark for analyzing the performance and scalability of rule-based systems at Web scale. OpenRuleBench partitions rule-based systems into five different categories: Prolog-based systems, Deductive databases, Rule engines for triples, Production and reactive rule systems, and Knowledge-base systems. The OpenRuleBench test suite includes a set of test derived from LUBM, adopting three out of its 14 queries. The three adopted queries are Query1, Query2 with high selectivity (i.e., each tuple of a join of 2 relation is joined with only a small number of tuples in the other relation) and Query3 with lower selectivity. Toward stream reasoning, Zang *et al.* [14] proposed SRBench, an RDF/SPARQL benchmark framework that covers dynamic reasoning in a stream-based context. However, the set of SRBench queries is restricted to sensor domain and is not clear to be easily and widely adapted like LUBM (e.g., to rule-based systems).

3 Systems Tested

In this work, we select the state-of-the-art reasoning engines for testing. The engines are widely used in many applications, and they are either meant for or adaptable for stream-based reasoning.

3.1 RDF Frameworks

Jena[2] is the state-of-the-art Java framework for building Semantic Web applications. Jena provides functionalities for triplet store, RDFS/OWL processing, reasoning and querying using SPAQRL. Jena reasoning module provides an interface for easy plugging in of external inference engines. OWLAPI[3] provides a high level application programming interface for creating and manipulating OWL Ontologies. Unlike Jena that support triple-based abstraction, OWLAPI focuses on a higher level of OWL abstraction syntax, the axioms. In this work, we study the two frameworks with an external reasoner component, Pellet.

[2] http://jena.apache.org/

[3] http://owlapi.sourceforge.net/

3.2 Pellet

Pellet[4] is an open source OWL-DL reasoner that supports incremental reasoning via two different approaches: i*ncremental classification* for ABoxes and *incremental consistency checking* for TBoxes. Incremental classification is a reasoning technique in Description Logic, which is used for incremental subsumption checking. Whereas this approach incrementally checks for changes in relations between classes in the ontology hierarchy (i.e., A *subClassOf* B or not), it requires expensive computational cost that makes Pellet work inefficiently when it comes to processing with continuous streams of data. *Incremental consistency checking* is used to instantly detect the consistency between triples in the KB. With that, Pellet removes the old triple facts that cause conflict with the newer triple facts and continuously look for inconsistencies in the KB without running all reasoning steps from scratch. This feature makes Pellet a potential candidate for stream reasoning, where the reasoner needs to be robust against the upcoming facts into the dynamic KB.

3.3 C-SPARQL

C-SPARQL is a new query language designed for data stream processing, based on the syntax of SPARQL with advanced features to support retrieving continuous results at runtime. The query range is applied on a specifiable sliding time window, with respect to a data source of one or multiple registered data streams combined with an optional static knowledge background. C-SPARQL *CONSTRUCT* predicate, inherited from SPARQL syntax, allows query results to be stored and registered as new streams on the fly, hence making it possible for C-SPARQL to describe production rules like *inference rules*. Figure 1 demonstrates the *transitivity rule* is constructed in C-SPARQL, the results are contained in a newly defined stream, namely, *inf*. Barbieri *et al.* [2] introduced further work on C-SPARQL in the domain of incremental reasoning, where each triple (both explicitly inserted and entailed) is tagged with an expiration time that describes the time the triple stays valid until it is retracted out of the inference engine. Incremental maintenance of materializations for a KB when facts change is controlled by a set of declarative rules.

3.4 Jess

Jess[5], Java expert system shell, is a popular rule based inference engine, built upon the Rete network, provides support for both backward and forward chaining. Because it is built upon the Rete Network, Jess supports incremental reasoning for forward inference. The backward chaining method in Jess requires a special declarations to inform the Jess engine that a rule has to be fired in order to acquire some facts. Facts in Jess are classified as ordered and unordered facts.

[4] http://clarkparsia.com/pellet/

[5] http://www.jessrules.com/

```
REGISTER STREAM inf COMPUTED EVERY
1s AS
PREFIX rdf:
<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl>
CONSTRUCT {?z ?p ?y}
FROM <static source>
FROM STREAM <some stream> [RANGE 1s STEP 1s]
WHERE { ?x ?p ?y .
?z ?p ?x.
?p rdf:type owl:TransitiveProperty
}
```

Fig. 1. An example of describing a transitive inference rule using C-SPARQL

Data structure to describe unordered facts in Jess are defined by templates. E.g. a template for RDF triple in Jess is : *(deftemplate triple (slot subject)(slot predicate)(slot object)*

3.5 BaseVISor

BaseVISor [15] is a forward chaining inference engine, based on the Rete algorithm, which is similar to Jess. However, BaseVISor is optimized specifically for triple processing, the system uses a triple-based data structure with binary predicates to express facts, hence simplifies the heavy work of pattern matching being done by the Rete engine. BaseVISOr provides functionalities to convey statements of n-ary predicate in RuleML and R-Entailment languages into its native raw triple structure, the BaseVISor language. All these characteristics make BaseVISor a good candidate for stream reasoning, where it has to deal with stream of RDF triples.

4 Methodology

LUBM provides an university ontology where 43 classes are defined, of which including *department*, *course*, *student*, *professor* etc. The benchmark generates triple data regardless of time. We extend LUBM by plotting its static ontology into a temporal semantic dimension, where we reconsider all possible triple data with respect to time. RDF Streams [16] are triples with additional timestamp annotation, as timestamp τ evolves monotonically in the stream.

$$\begin{gathered} ... \ (< subj_i, pred_i, obj_i >, \tau_i) \\ (< subj_{i+1}, pred_{i+1}, obj_{i+1} >, \tau_{i+1}) \ ... \end{gathered}$$

RDF Streams generated from our modified data generator of LUBM followed Barbieri *et al.*'s definition. The only difference we made is, as the KB is in a university domain, we choose a semester as the time granularity. This allows triples in the KB to retain their semantic meaning, adding the semester period when they are valid. By this, the timestamp of triple facts still follows a monotonic

Object Property	Temporal Type
has a degree from	static
has as a member	near-dynamic
has as a research project	near-dynamic
has as an alumnus	static
is a TA for	dynamic
is about	static
is affiliated with	near-dynamic
is being advised by	dynamic
is taking	dynamic
publishes	static
teaches	dynamic
was written by	static

Table 1. List of LUBM object Properties and their temporal types

evolution and are semantically validated by academic semesters. As the consequence, there is a batch of facts corresponding to every semester. In this semester timespan, a semester can be considered as a *time slice* where facts in the newest time slice are all relevant. We dont use finer-grain time granularity to a semester (e.g., day or week) because this time slide length ensures a good proportion of dynamic data in a university data-generated context of LUBM.

... (<Department0.University0/GraduateStudent31, ub:takesCourse , Department1.University0/GraduateCourse1>, *semester0*) ...

As the time evolves, the benchmark re-generates university data *recursively* as complete batches (complete *university* data) of information. After each recursion, *dynamic* data in the KB of the previous semester are considered expired and must be retracted and replaced by newer data (of the new semester). In our system architecture described in Figure 2, the RDF Handler module only accepts triples (generated from LUBM ABox) with temporal annotation after the first recursion. This mechanism guarantees the inference engine does not have to process duplicate *static* data (already there in the first round). Table 1 provides a list of all object properties provided in the LUBM Ontology. The object properties are classified with regards to the change probability over time of the according objects in its range. We heuristically define three distinct predicate classes, namely static, near-dynamic and dynamic. Static is labeled for those predicates which reflect facts that are true for the whole timespan, while dynamic indicates those predicates that the fact validity changes constantly for every time slice or every window of time slices in the case of near-dynamic. For example, *has a degree from* refers to a constant fact where the object class of *ub:University* is labeled as static, while a subject instance with *teaches* has a more subject to change object instance (e.g., a teacher is assigned to teach different subjects in different semester). Therefore, with respect to the time dimension of semester unit, *teaches* is marked as dynamic. The temporal predicates (dynamic and near-dynamic decides the proportion of dynamic data generated in each recursion, and hence, the complexity of the temporal data (towards reasoning engines).

Stuckenschmidt *et al.* [6] described a conceptual view of a stream reasoner where data streams are processed from high frequent fine grain data streams into low frequent aggregated events. They also proposed a desired architecture for stream reasoners to deal with the trade-of between its methods' complexity and the frequency of data stream (e.g., description logics are not able to deal with high frequency data streams). On the lower level, raw data stream are fed into a stream data management system that warps the data stream into a virtual RDF data models, before being passed to higher levels of logic programs and description logic. In our extended LUBM, we reduce the trivial work for tested reasoners by omitting the bottom level of the cascading hierarchy of stream reasoning. In particular, the benchmark provides a *triple buffer* (in RDFHandler in Figure 2), where data streams generated directly from LUBM are processed and from that RDF triples are consolidated and streamed into the reasoner.

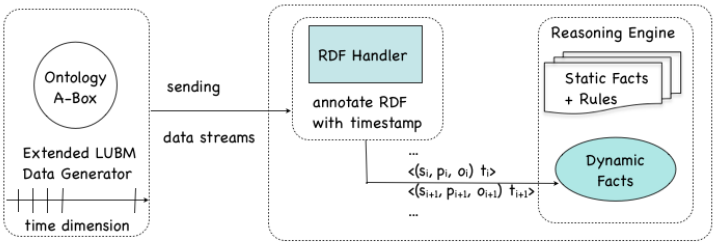

Fig. 2. Extended LUBM for semantically temporal RDF Stream

In the communication module, we replace the original way of data transfering in traditional LUBM (which RDF data are serialized into files and the files then will be read by the reasoner), by a stream-based protocol. Here, LUBM data generator (the RDFWriter module in particular), is improved to establish a TCP/IP socket connection with experimented reasoners. To make it uniformly work, we introduce a data buffer that temporarily stores stream data from LUBM, RDFHandler, where RDF stream is parsed into RDF triples and these triples will be then fed into the tested reasoner via its API interface for processing.

For stream reasoning, inference rules for rule-based reasoning engines are redesigned. Table 2 contains a list of the most important inference rules and introduces a time-to-last $\triangle_t$ which describes the expiration time of the conclusions that derive from these first-order rules. Time-to-last $\triangle_t$ has a lower bound of one semester, the time unit of the time dimension, and has no restriction in the upper bound as a derived fact can last for several semesters. In our extended LUBM, for consistency purpose, a time-to-last $\triangle_t$ of an inference rule is defined as the minimum number of time units for facts that constitutes the first part of

Temporal Inference Rule	**Description**
transitive	(s_0, p_0, o_0) (o_0, p_0,o_1) (p_0, rdf:type, owl:TransitiveProperty)) ⇒ (*assert* ((s_0, p_0, o_1), $\triangle_t$))
subclass-type transitive	(s_0, rdfs:subClassOf, o_0) (o_0, rdf:type, o_1) ⇒ (*assert* ((s_0, rdf:type, o_1), $\triangle_t$))
subproperty-type transitive	(p_0, rdfs:subPropertyOf, p_1) (s_0, p_0, o_0) ⇒ (*assert* ((s_0, p_1, o_0), $\triangle_t$))
symmetry	(p_0, rdf:type, owl:SymmetricProperty) (s_0, p_0, o_0) ⇒ (*assert* (o_0, p_0, s_0), $\triangle_t$))
inverse	(p_0, owl:inverseOf, p_1) (s_0, p_0, o_0) ⇒ (*assert* (o_0, p_1, s_0), $\triangle_t$))
equivalent	(o_0, owl:equivalentClass, o_1) (s_0, rdf:type, o_0) ⇒ (*assert* (s_0, rdf:type, o_1), $\triangle_t$))

Table 2. List of main temporal inference rules for LUBM queries

the inference rules stay in the system until one of them is removed out of the KB (so that, the *outdated* inferenced fact is also removed).

5 Experiments

All experiments were conducted under Linux 2.6.x 64 bit, on top of a server computer with an Intel(R) Xeon(R) E7520 1.87GHz processor and 80GB memory. The Java engine was OpenJDK 1.6.0_24. We experimented with four different reasoning engines, two of them are RETE-based (Jess, BaseVISOr), one description logic-based (Pellet, together with Jena and OWLAPI) and one is the engine of C-SPARQL. 14 LUBM queries are re-used to assess the engine performances in a stream-based approach. We measured the loading time, query response time and the completeness and soundness of the returned results from the tested engines. For simplicity, we chose *takescourse* as the only *temporal* predicate (adding other predicates does not significantly change the complexity of the system). Hence, there is approximately 10% of the generated data are dynamic (measured by the proportion between the numbers of facts with *takescourse* predicate and the total number of facts). We first experimented with Jess engine for the load+inference (loading triple and inference on the fly) of Stream LUBM (1,0,5),which is LUBM (1,0) running over 5 semester. The results surprisingly show that Jess engine, based on the RETE algorithm (ideal for real-time pattern matching [17]), gave a low performance for this test. It took Jess 419,82s for the load+inference time and 414,14s for answering LUBM Query 14 with 5403 results. For the same dataset, BaseVISor, a similar RETE-based engine with optimization on triple processing, outperforms Jess, spending 11,33s for loading time and 0,14s for Query 14. Because of this, we decided to omit further results of Jess in this paper.

BaseVISor, Pellet+Jena and Pellet+OWLAPI are fed with data from different range. Figure 3 shows the response time on LUBM queries of BaseVISor for Query 5, Query 6, Query 13 and Query 14. In general, BaseVISor performed well

in the incremental reason fashion as computation is reduced to minimal after the first or second semester. We chose Query 14 for most of the test scenarios (although it has the lowest complexity among LUBM 14 queries) because the query guarantees a large result set that is in propositional to the size of the generated LUBM data. It can be seen that the accumulation of computation as the engine takes more time on the first semester and takes significantly less time on the following semesters. Figure 4 indicates the time BaseVISor uses to response to Query 14, the longest query time among the LUBM queries for BaseVISor over the dataset range from LUBM (1,0,5) to (50,0,5) in 5 semesters. The chart shows the incremental behaviour of the RETE engine after semester 1, as the computation time needed for the next semesters are similar regardless of the size of the dataset, and is closed to zero. The results (detailed in Table 3) indicate BaseVISor's efficiency for stream reasoning.

Because Pellet+Jena and Pellet+OWLAPI are respectively founded on Jena and OWLAPI for RDF processing, RDF triples cannot be directly fed into both system. The statements of Jena and axioms of OWLAPI require elements of RDF triples to be pre-classified as resources and properties of the predefined ontology. For example, (ub:GraduateStudent0 ub:takesCourse http://www.Department0.University0.com/Course0) RDF triple, *ub:GraduateStudent0* must be recognized as an instance of *ub:GraduateStudent* class, similarly for the predicate and object. Another medium module for triple classification is added for the pre-processing. Pellet+Jena and Pellet+OWLAPI, are based on the Description Logic reasoning engine of Pellet, show less significant figures than BaseVISor in query time required to answer the 14 LUBM queries, however both take much less time for data loading as tested with LUBM (5,0,5), (10,0,5), (20,0,5) and (50,0,5) in Figure 7. Pellet+Jena and Pellet+OWLAPI performance on incremental reasoning are measured as well as BaseVISor in Figure 5 and the results show that the BaseVISor outperforms the two DL-based engines. We also provide the numerical figures in Table 3 where the systems are experimented with Query 6, Query 13 and Query 14 and different settings for references.

We have also tested with the engine of C-SPARQL, an emerging and promising engine for stream RDF processing. With this engine, the data stream generated from our extended LUBM is registered as the only stream source. We have created the inference rule set (as described in Table 2) using *CONSTRUCT* predicate, in the same analogy to what we described in Figure 1. However, the released package of C-SPARQL[6] does not yet fully support stream reasoning and as C-SPARQL returned results continuously over time, which requires a different approach of querytime measurement to the rest of the tested systems, thus we decided to present only the engine's loading time. As shown in Figure 7, C-SPARQL takes the least time to load in the data.

[6] `http://streamreasoning.org/larkc/csparql/CSPARQL-ReadyToGoPack-0.8.zip`

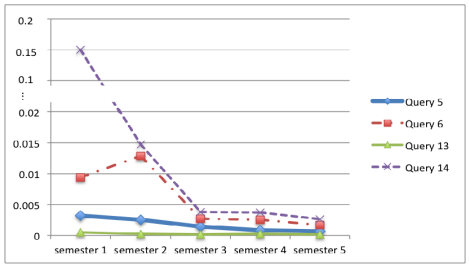

Fig. 3. BaseVISor query-time for LUBM queries for extended LUBM (1,0,5), which is LUBM(1,0) over 5 semesters

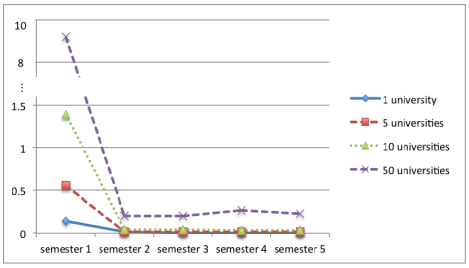

Fig. 4. BaseVISor query-time for LUBM query 14 for extended LUBM (1,0,5), (5,0,5), (10,0,5) and (50,0,5)

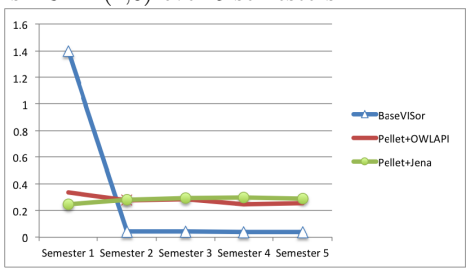

Fig. 5. BaseVISor, Pellet+OWLAPI and Pellet+Jena Query 14 time for extended LUBM (10,0,5)

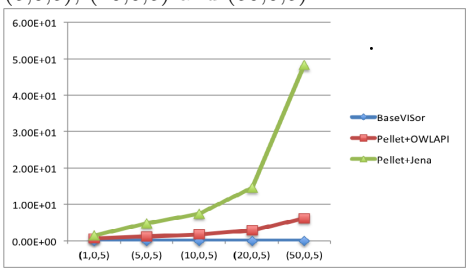

Fig. 6. Query time for extended LUBM (1,0,5), (5,0,5), (10,0,5), (20,0,5) and (50,0,5)

6 Conclusion

In this paper, we have introduced an extension of the well-known benchmark (LUBM) for reasoning engines over *static* dataset, to make it suitable for testing stream-based systems. The extension preserves the semantic of the LUBM ontology while adding a time dimension (of semester unit) to the KB. We run experiments on full and partial stream reasoning supported engines (i.e., BaseVISor, Pellet and C-SPARQL)using our benchmark. The results reflect the capabilities of each system in stream reasoning context, where BaseVISor outperforms other engines in most of the test cases.

For future work, we will focus on three improvements. First, our current approach is limited to control the complexity with time (e.g., increase the complexity the system must handle in the next semester time units). Secondly, we want to extend the benchmark to feature a more direct measurement of the time saved by incremental reasoning (e.g., how much computation is saved after one time unit). And lastly, we want to extend the benchmark to generate inconsistent facts (e.g., from different sources) with a parameterized rate, in order to closely simulate noisy real-world stream-based applications.

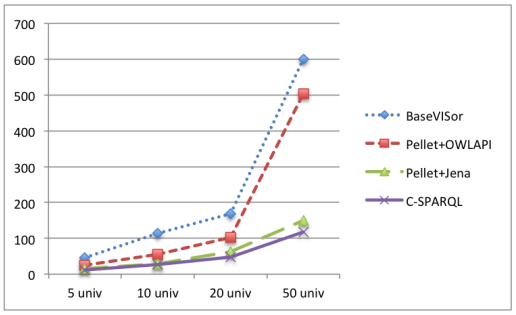

Fig. 7. Loading time for extended LUBM (5,05), (10,0,5), (20,0,5) and (50,0,5)

7 Acknowledgement

This work is funded by BMBF under the project ASEV.

References

1. Della Valle, E., Ceri, S., van Harmelen, F., Fensel, D.: It's a streaming world! reasoning upon rapidly changing information. Intelligent Systems, IEEE **24**(6) (2009) 83–89
2. Barbieri, D.F., Braga, D., Ceri, S., Della Valle, E., Grossniklaus, M.: Incremental reasoning on streams and rich background knowledge. In: Proceedings of the 7th international conference on The Semantic Web: research and Applications - Volume Part I. ESWC'10, Berlin, Heidelberg, Springer-Verlag (2010) 1–15
3. Zeng, D., Chen, H., Lusch, R., Li, S.H.: Social media analytics and intelligence. IEEE Intelligent Systems **25**(6) (2010) 13–16
4. Della Valle, E., Ceri, S., Fensel, D., Harmelen, F., Studer, R., eds.: Commonsense Spatial Reasoning about Heterogeneous Events in Urban Computing, CEUR Workshop Proceedings (2009) First International Workshop on Stream Reasoning: SR2009.
5. Heintz, F., Kvarnstrm, J., Doherty, P.: Stream-based reasoning in dyknow. In: Proceedings of the Dagstuhl Workshop on Cognitive Robotics. (2010)
6. Stuckenschmidt, H., Ceri, S., Valle, E.D., van Harmelen, F.: Towards expressive stream reasoning. In Aberer, K., Gal, A., Hauswirth, M., Sattler, K.U., Sheth, A.P., eds.: Semantic Challenges in Sensor Networks. Number 10042 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany (2010)
7. Guo, Y., Pan, Z., Heflin, J.: Lubm: A benchmark for owl knowledge base systems. J. Web Sem. **3**(2-3) (2005) 158–182
8. Prud'hommeaux, E., Seaborne, A.: SPARQL query language for rdf. W3C Recommendation **4** (2008) 1–106
9. Bock, J., Haase, P., Ji, Q., Volz, R.: Benchmarking OWL Reasoners. In van Harmelen, F., Herzig, A., Hitzler, P., Lin, Z., Piskac, R., , Qi, G., eds.: Proceedings of the ARea2008 Workshop. Volume 350., http://ceur-ws.org, CEUR Workshop Proceedings (June 2008)

LUBM Query	Dataset	Engines	Semester 1	Semester 2	Semester 3	Semester 4	Semester 5
Query 6	LUBM(1,0,5)	**BaseVISor**	**0.0094**	**0.0128**	**0.0027**	**2.56E-03**	**1.70E-03**
		Pellet+OWLAPI	0.2904	0.0695	0.0541	0.0678	0.0657
		Pellet+Jena	0.3250	0.2148	0.0657	0.0649	0.0580
	LUBM(10,0,5)	**BaseVISor**	**0.0752**	**0.0387**	**0.0376**	**0.0417**	**0.0421**
		Pellet+OWLAPI	1.0790	1.3778	0.9639	1.2473	0.9325
		Pellet+Jena	0.8824	0.9220	0.9187	0.9683	0.9381
	LUBM(50,0,5)	**BaseVISor**	**0.2120**	**0.2186**	**0.2876**	**0.2920**	**0.2420**
		Pellet+OWLAPI	2.6468	5.0834	4.7486	7.0057	5.010
		Pellet+Jena	2.8227	5.8366	3.9432	3.9606	4.6816
Query 13	LUBM(1,0,5)	**BaseVISor**	**4.86E-04**	**2.59E-04**	**1.64E-04**	**2.67E-04**	**1.32E-04**
		Pellet+OWLAPI	0.6151	0.1142	0.099	0.1763	0.0129
		Pellet+Jena	1.4611	0.1435	0.0579	0.0032	0.0419
	LUBM(10,0,5)	**BaseVISor**	**4.05E-04**	**3.85E-04**	**3.56E-04**	**4.10E-04**	**3.31E-04**
		Pellet+OWLAPI	1.8288	1.8845	1.9835	2.2229	1.8484
		Pellet+Jena	7.4055	1.099	0.8322	0.7990	0.8555
	LUBM(50,0,5)	**BaseVISor**	**7.34E-04**	**5.39E-04**	**4.64E-04**	**6.17E-04**	**4.95E-04**
		Pellet+OWLAPI	6.2388	9.9391	12.0995	10.4035	9.8907
		Pellet+Jena	48.2975	7.0330	6.1593	5.3822	5.0759
Query 14	LUBM(1,0,5)	**BaseVISor**	0.1429	**0.0140**	**0.0037**	**0.0036**	**0.0026**
		Pellet+OWLAPI	**0.0447**	0.0318	0.0140	0.0145	0.0139
		Pellet+Jena	0.0641	0.0329	0.0172	0.0175	0.0146
	LUBM(10,0,5)	**BaseVISor**	1.397	**0.0400**	**0.0400**	**0.0353**	**0.0360**
		Pellet+OWLAPI	0.3343	0.2774	0.2844	0.2437	0.2529
		Pellet+Jena	**0.2437**	0.2789	0.2945	0.2952	0.2878
	LUBM(50,0,5)	**BaseVISor**	9.3391	**0.2015**	**0.2036**	**0.2688**	**0.2286**
		Pellet+OWLAPI	**0.9989**	1.5743	1.3088	1.3499	1.6095
		Pellet+Jena	1.0327	1.5403	1.7234	1.3564	1.6448

Table 3. Query time evolves over semesters for LUBM Queries on extended LUBM (1,0,5), (10,0,5) and (50,0,5) dataset in seconds

10. Ma, L., Yang, Y., Qiu, Z., Xie, G., Pan, Y., Liu, S.: Towards a complete owl ontology benchmark. In: Proceedings of the 3rd European conference on The Semantic Web: research and applications. ESWC'06, Berlin, Heidelberg, Springer-Verlag (2006) 125–139
11. Schmidt, M., Hornung, T., Lausen, G., Pinkel, C.: Sp2bench: A sparql performance benchmark. CoRR **abs/0806.4627** (2008)
12. Bizer, C., Schultz, A.: The berlin sparql benchmark. Int. J. Semantic Web Inf. Syst. **5**(2) (2009) 1–24
13. Liang, S., Fodor, P., Wan, H., Kifer, M.: Openrulebench: an analysis of the performance of rule engines. In: Proceedings of the 18th international conference on World wide web. WWW '09, New York, NY, USA, ACM (2009) 601–610
14. Zhang, Y., Minh Duc, P., Corcho, O., Calbimonte, J.P.: Srbench: A Streaming RDF/SPARQL Benchmark. In: Proceedings of International Semantic Web Conference 2012. (November 2012)
15. Matheus, C., Baclawski, K., Kokar, M.: Basevisor: A triples-based inference engine outfitted to process ruleml and r-entailment rules. In: Rules and Rule Markup Languages for the Semantic Web, Second International Conference on. (2006) 67–74
16. Barbieri, D.F., Braga, D., Ceri, S., Della Valle, E., Grossniklaus, M.: C-sparql: Sparql for continuous querying. In: Proceedings of the 18th international conference on World wide web. WWW '09, New York, NY, USA, ACM (2009) 1061–1062
17. Forgy, C.L.: Rete: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence **19**(1) (1982) 17 – 37