Technical Report: LUBM Performance Results for INSTANS October 2015

Mikko Rinne
Department of Computer Science,
Aalto University, School of Science,
Finland
mikko.rinne@aalto.fi

Esko Nuutila
Department of Computer Science,
Aalto University, School of Science,
Finland
esko.nuutila@aalto.fi

ABSTRACT

This report contains the performance results of the INSTANS RDF event stream processor for the LUBM benchmark, executed in October 2015. The tests incorporate RDF(S) to ρdf, D*, P-entailment and OWL 2 RL entailment regimes as well as test-optimised variants. Performance is also compared to established non-streaming SPARQL platforms.

1. EXPERIMENTAL SETUP

To compare the performance of complete entailment regimes with customised sets of rules and the performance of event-like processing with static processing, as well as the performance of INSTANS with other platforms supporting reasoning, a benchmark was needed. We chose LUBM[1] [1] because it:

- has been successfully used in benchmarking for 10+ years
- has queries with and without reasoning requirements
- is not based on streaming data and can therefore be used on platforms without streaming support
- features independent segments (departments), which can be used to simulate a data stream

Outside the RDF stream processing scope there are many established reasoning and SPARQL query processing platforms.

For the performance comparison we used *Jena*[2] as a well-known platform with an open and configurable materialisation-based reasoner and *Stardog*[3] as an example of a high-performance tool with query rewriting.

[1]http://swat.cse.lehigh.edu/projects/lubm/
[2]https://jena.apache.org/
[3]http://stardog.com/

Generator version *UBA 1.7* was used to generate data for 1, 5 and 10 universities[4]. Both the data and the benchmark ontology were converted and packaged to single turtle[5] files using the rdf2rdf[6] converter. Query syntax was aligned with SPARQL 1.1, otherwise the queries were not touched.

INSTANS v. 0.3.0.0 was compiled with a heap size of 32768M and used from the command line. Stardog version 3.1.4 with default settings ("SL" reasoner) was used from the command line. A *reasoner-jena*[7] wrapper Scala[8] v. 2.11.7 application was created to support selecting the reasoner from command line and timing the execution, running Jena v. 3.0.0. Maximum heap size was set to (Xmx) 32G.

All experiments were executed on a MacBook Pro with a 2.7 GHz Intel Core i5 processor and 16 GB of 1867 MHz DDR3 memory running OS X Yosemite 10.10.5. All speed tests were executed four times, the first run ignored as warm-up and the median value of the three remaining runs recorded as the result. For INSTANS and Jena the timer was started before opening the data file and stopped at the end of iterating through the results. Stardog pre-reads files into a database in a separate step before processing. We used the "real" value of the unix-command "time" to measure the time used by Stardog. An average delay of running Stardog with no data[9] was subtracted as overhead and the time of reading the data into the database (Table 1) was added. It should, however, be noted that since Stardog parses the data into the database only once, the run-time experience for the end-user is 3-6s faster than indicated in the results. Stardog suffered from some stability problems, with the server deterministically halting on the 3rd execution of Q6 through the batch. Re-starting the Stardog server for each trial stabilised the situation[10].

2. PERFORMANCE RESULTS

The LUBM web page provides reference query answers for one university. Even though LUBM is not an RL ontology, as it contains existentially quantified axioms, it was verified that all the platforms and qualifying sets of rules and reason-

[4]seed 0, index 0
[5]http://www.w3.org/TR/turtle/
[6]http://www.l3s.de/~minack/rdf2rdf/ v. 1.0.1-2.3.1
[7]https://github.com/aaltodsg/reasoner-jena
[8]http://www.scala-lang.org/
[9]~2.2s with reasoning, ~1.5s without
[10]INSTANS and Jena also re-start completely for each execution

Table 1: Number of triples and loading time measured by Stardog (U = University).

Set	Triples	Stardog load time [s]
1U	100,545	1.616
5U	624,534	3.086
10U	1,272,577	6.261

ers perfectly reproduce the reference answers. A list of all 14 queries together with the minimum set of rules as OWL 2 RL names as well as the compliant reasoning frameworks for INSTANS and Jena are listed in Table 2. Q1-Q3 and Q14 use no entailments. A maximum of three rules per query are required, and only eight rules are sufficient to pass all queries. The execution speed of each framework (reasoner or regime) was verified using five universities, the lists in the table are ordered fastest first. Entries in parenthesis would theoretically be compliant but failed to complete in practice.

Table 2: LUBM queries, required rules with OWL 2 RL names, qualifying entailment regimes and Jena reasoners (fastest first, ()=did not complete in test).

Query	Rules	INSTANS	Jena
Q1-Q3	none		
Q4	cax-sco	RDFS ρdf D* (OWL2RL)	RDFSSimple RDFS OWLMicro OWLMini OWL
Q5	prp-dom prp-spo1 cax-sco	RDFS ρdf D* (OWL2RL)	RDFSSimple RDFS OWLMicro OWLMini OWL
Q6 - Q10	cls-int1 cax-sco scm-svf1	(OWL2RL)	OWL (Q7)
Q11	prp-trp	P-entail (OWL2RL)	OWLMicro OWLMini OWL
Q12	cls-int1 cax-sco cls-svf1	(OWL2RL)	OWL
Q13	prp-dom prp-spo1 prp-inv1	D*+P-ent (OWL2RL)	OWLMicro OWLMini OWL
Q14	none		

Performance results for queries requiring no reasoning are shown in Figure 1. INSTANS completed Q2 with one university (1U), but exhausted the heap with 5U. The high memory consumption is due to a large number of candidate solutions, which the event-based memory handling does not help with. Jena Q2 performance is also comparatively low and decreasing at 0.67 (5U) and 0.36 (10U) ktriples/s. Comparing the 5U and 10U cases over the other queries we see that INSTANS performance is stable already at 5U, whereas Jena and Stardog continue to accelerate from 5U to 10U. The optimised reasoner of Stardog shows very minor impact on performance. Stardog with reasoner performs 12x-14x faster than event-based INSTANS without reasoning (10U case), indicating an upper limit for what could be achieved on INSTANS using a reasoner based on query rewriting (assuming no delay from the reasoner).

The impact of event-based memory-handling on INSTANS must be analysed over all test cases. Figure 2 shows the INSTANS memory allocation sampled once per second for both event-based and static memory handling approaches using the OS X system utility *top* running Q9 (1U). The static approach exhausted heap memory and did not complete, whereas the more dynamic event-based approach shows much better memory stability. Without entailments (Figure 1) the static approach consistently performs ∼1.3x faster than event-based (5U), with limited entailment regime queries (Figure 3) static was 3.9x-6.8x faster (5U opt). In other queries (Figure 4 and Figure 5) the static approach collapses: Q7 and Q9 ran out of memory, Q12 did not complete in 44 hours (<3.9 triples/s). For the ones that did complete (Q6, Q8, Q10) the situation reversed and event-based was measured to be ∼29x faster in all three cases. This shows how different aspects of memory pressure impact INSTANS performance: in simpler cases the delay of erasing the main graph after each department (event-based approach) is significant and the static approach is faster, whereas in the more complex cases the additional in-memory bindings in the static case collapse performance and the event-based approach performs better.

Without entailments (Figure 1) Jena was measured 9.3x-9.9x faster than INSTANS (5U Event). Moving to queries with smaller entailment regimes (Figure 3) using the simplest and fastest regimes available on both INSTANS and Jena (Table 2) producing the correct results, Jena performed 11.7x-102x faster than INSTANS. Using the best measured performance (static case) with optimised rules INSTANS performance improves 1.4x-16x, leaving Jena 5.1x-8.3x faster and Stardog 10x-18x faster.

The situation changes dramatically with the queries requiring the most complete *OWL* reasoner on Jena. Q7 failed to complete, other queries took 1-27 hours on a single university. With 5U Q6 did not complete a single iteration in 59 hours (less than 3 triples/s). OWL 2 RL on INSTANS (using the static approach) did not complete any query either. Using only the necessary rules and event-based processing, INSTANS successfully completed all remaining queries both for 5U and 10U (Figure 4 and Figure 5). Stardog performed 22x-447x faster than INSTANS (10U Event). Both Jena[11] and Stardog[12] support rule customisation and therefore the same optimised sets of rules as the ones used with INSTANS could be generated also for these tools. As the study focuses on the ease of customisation using SPARQL vs. pre-existing reasoner frameworks, not a head-to-head comparison of the platforms, no rule customisation on Jena or Stardog was done.

3. CONCLUSIONS

[11] custom syntax
[12] SWRL: http://www.w3.org/Submission/SWRL/

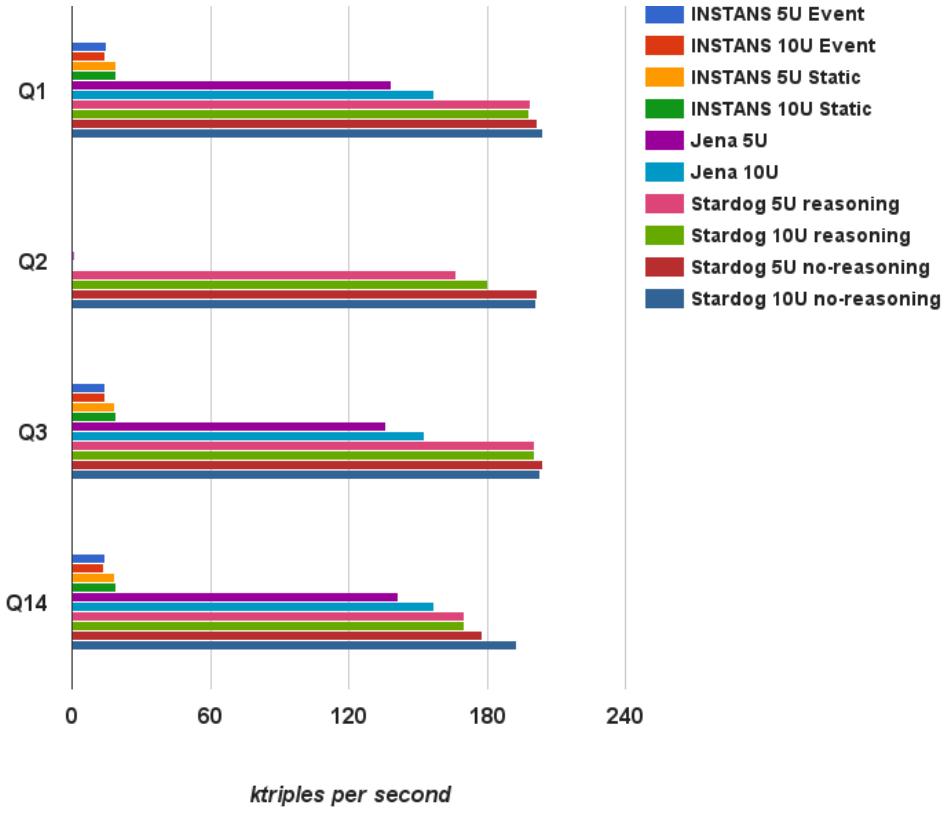

Figure 1: Execution speed results for queries requiring no entailment (ktriple = 1,000 triples).

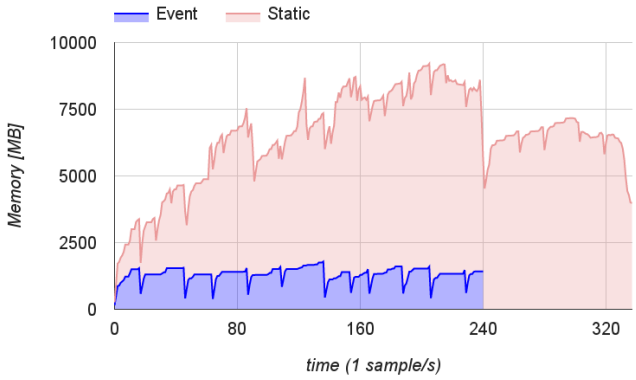

Figure 2: INSTANS memory allocation for 1 university in Q9.

We verified the successful reproduction of LUBM reference answers with all queries both for our rule implementation on INSTANS and on the comparison platforms. We tested which reasoners on Jena and which entailment regimes on INSTANS produced correct results for each query, and their relative performance. With queries utilising limited rule sets Jena with the minimum applicable reasoners performed 11x-102x faster than INSTANS. Using only the necessary rules on INSTANS improved performance 1.4x-16x, leaving Jena 5.1x-8.3x faster. In queries requiring the Jena *OWL* reasoner Jena performance collapsed, whereas INSTANS completed all queries also with 10 universities using minimum sets of rules per query.

Materialisation as a method of reasoner implementation consumes capacity (whether the inferred triples are stored in memory or database). This does not match well with stream processing, where data is assumed to come as an infinite stream. Despite that the presence of independent data segments such as time windows, events or university departments (LUBM) often enables construction of a case-specific solution, which separates the static data (including the ontology and ontology-based materialisations) from the dynamic data by using named graphs. A solution for LUBM was tested. The memory consumption was shown to be clearly decreased and stabilised, observed best in the case of LUBM Q9, where a single university crashed 32GB of heap memory without the event-based solution, whereas 10 universities passed without problems with the solution activated. We also observed that the performance impact of the event-approach on the INSTANS platform was case-specific. In simpler cases delay due to deletion operations slows down the event-based approach, whereas data accumulation in more complex cases can dramatically slow down performance of the static approach. The event-based approach was also demonstrated to have significantly better stability.

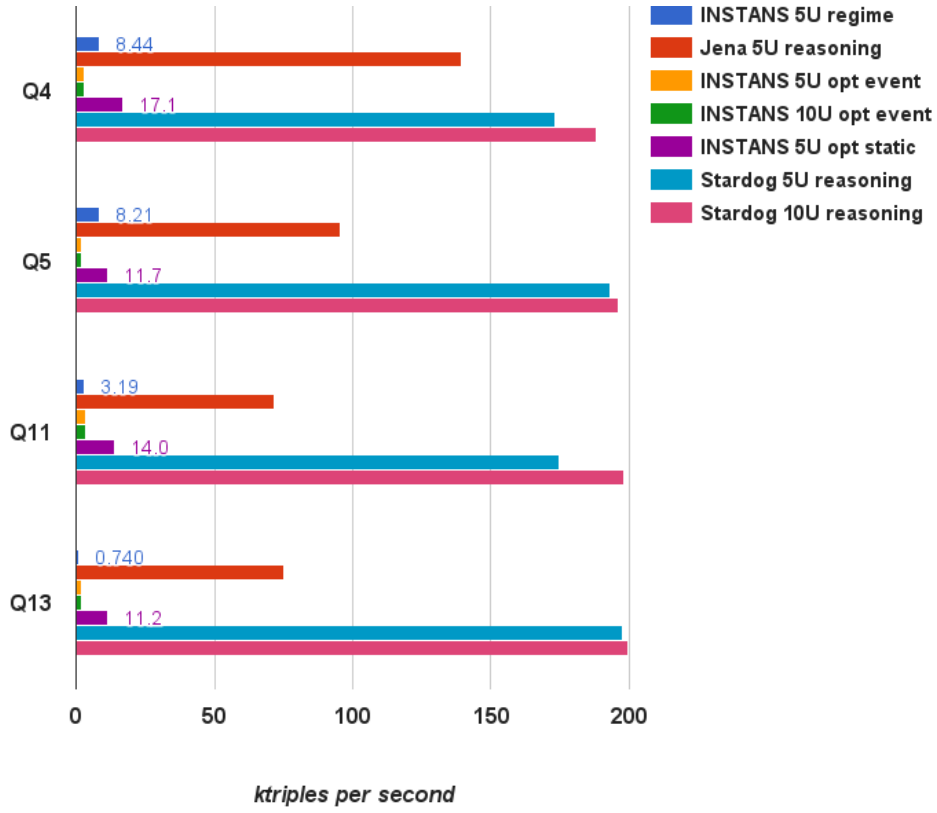

Figure 3: Execution speed results for queries requiring small entailment regimes.

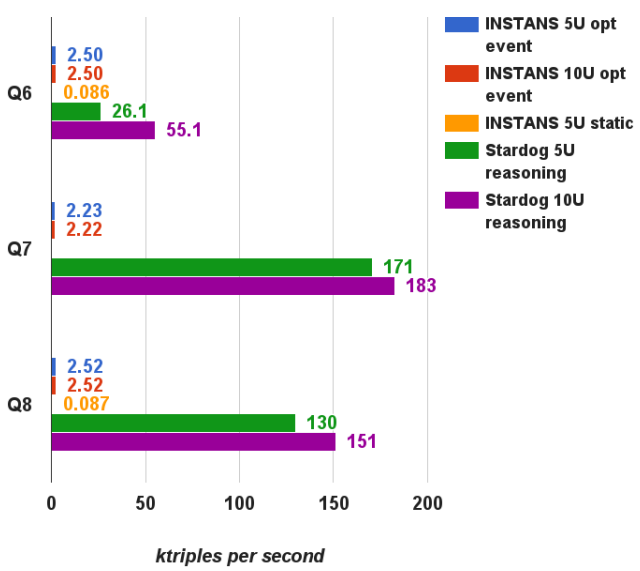

Figure 4: Execution speed results for Q6-Q8.

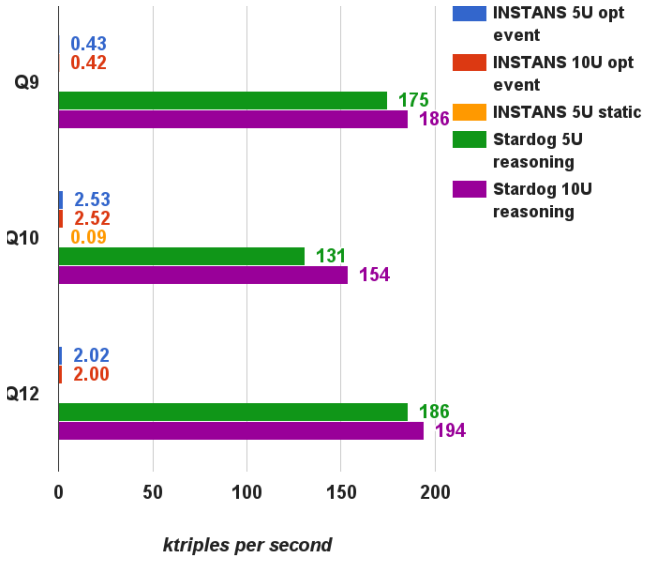

Figure 5: Execution speed results for Q9, Q10 and Q12.

Stardog with an optimised query rewriting reasoner demonstrated clearly superior performance to either one of the materialisation-based reasoning platforms. Comparing Stardog performance with reasoner to INSTANS in queries not utilising reasoning shows Stardog to perform 12x-14x faster. If no other optimisations are performed, this gives the upper limit for what could be achieved with a zero-delay query rewriting component on INSTANS, assuming no other optimisations. As a future task a query rewriting reasoner should be tested over INSTANS.

Acknowledgments

This work has been carried out in the TrafficSense project funded by Aalto University.

4. REFERENCES

[1] Guo, Y., Pan, Z., Heflin, J.: An Evaluation of Knowledge Base Systems for Large OWL Datasets. In: ISWC 2004. vol. 3, pp. 274–288 (2004)