$\mathcal{H}$eaven Test Stand: towards comparative research on RSP engines

Riccardo Tommasini, Emanuele Della Valle, Marco Balduini, Daniele Dell'Aglio

DEIB, Politecnico of Milano, Milano, Italy
riccado.tommasini@polimi.it, emanuele.dellavalle@polimi.it,
marco.balduini@polimi.it, daniele.dellaglio@polimi.it

Abstract. The benchmarking of window-based RDF Stream Processing (RSP) engines has recently attracted the attention of the Stream Reasoning community. Solutions like LSBench, SRBench and CSRBench tried to fulfill the need of shared practices for RSP engine evaluations. However, an infrastructure for the systematic comparison of existing systems is still missing. In this paper, we propose the requirements and an architecture for $\mathcal{H}$eaven Test Stand, a facility to foster Systematic Comparative Research Approach (SCRA) for window-based RSP engine. $\mathcal{H}$eaven allows to design and systematically execute repeatable, reproducible and comparable experiments. As further contribution a working implementation of $\mathcal{H}$eaven Test Stand is released as open source.

1 Introduction

A Systematic Comparative Research Approach [4] (SCRA) is commonly used in those research fields where formulating hypothesis to test is extremely hard, due to the complexity of the research subjects. SCRA main advantage consists into reducing cross-case studies complexity by representing the subjects as combinations of known properties and highlight differences and similarities.

Social sciences, like sociology or economy, are extremely relevant fields [14] where comparative analysis is exploited to understand the subjects, bringing together the strengths of qualitative approaches and quantitative ones (e.g, in situation X, the $Y\%$ of population A grows the $Z\%$ faster than the rest).

Due to the multiple technology concepts involved, Stream Reasoning (SR) [6] - a research field that couples Information Flow Processing approaches [5] with reasoning techniques - clearly demands for SCRA. SR feasibility is already proven; many RDF Stream Processing (RSP) engines - systems able to cope with semantically annotated data flows - populate the state-of-the-art [12].

The relevant challenge for the entire community[1] is now the RSP engine evaluation. To the RSP benchmarking extent are currently available RDF Streams, Ontologies and Continuous queries [7, 11, 16]. The performance measure sets comprise query language expressiveness [11, 16], scalability [11], throughput [11], query results' mismatch [11] and correctness [7]. [7, 11] provide also a preliminary

[1] http://www.w3.org/community/rsp/

testing facility, but *an infrastructure for systematically compare the RSP engine execution under controlled conditions is still missing.*

In related research areas like databases, the results of the evaluation is know given a schema of the data , the data, and a query. Therefore, there is no need to study the systems dynamics at once. Unfortunately, this is not valid for RSP engines where the execution semantic is different from system to system [2]. In [7] problem of modeling the system execution semantics was faced for window-based, in memory RSP engines; it depends on the relation between *RDF Streams* [9]; *continuous extensions of SPARQL* [3] and *streaming-adapted reasoning techniques* [9]. However, even when the model is completely available, the intrinsic complexity of the RSP engine is still high and it requires to analyze the dynamics at once. An SCRA is preferable and, thus, our research question is *How to enable a systematic comparative research approach for RSP engines?*

In this paper we answer such research question presenting $\mathcal{H}$eaven Test Stand an infrastructure to enable experiment design and execution for RSP engines.

Outline: Section 2 and 3 present respectively $\mathcal{H}$eaven Test Stand requirements and architecture. Section 4 provides an example of experiment design an some details regarding an open source implementation of the $\mathcal{H}$eaven Test Stand[2]. Finally, Section 5 comes to conclusion and presents the future works.

2 $\mathcal{H}$eaven Requirements

Our aim is studying the RSP engine dynamics, that requires on-line analysis of the engine behavior. SCRA fosters cross-case analysis and, thus, we need to guarantee the experimental conditions to be *repeatable*, *reproducible*, and *comparable* and also to describe the variables involved in the evaluation.

From the aerospace engineering we borrow the notion of *engine test stand*, a facility for to design and systematically execute experiments over engines. With a test stand, it will be possible to collect performance measurements during the engine execution and enable post-hoc evaluations. In the following we state the requirements for an RSP engine Test Stand, namely $\mathcal{H}$eaven Test Stand ($\mathcal{H}$TS).

Experiments *Reproducibility* requires $\mathcal{H}$TS to guarantee: (i) engine compatibility (R.1), to allow the comparison of experiment results; (ii) data independence (R.2), to allow the system users to choose any relevant ontologies (R.2a) and to describe how the input data stream is defined (R.2b) (e.g. dataset/API, data model like RDF Stream etc.); (iii) query independence (R.3), to allow the system users to define and register a set of relevant queries from their domain of interest.

Compatibility with the RSP benchmarking state-of-the-art is crucial for $\mathcal{H}$TS relevance, thus, $\mathcal{H}$eaven design must be extensible at software level (R.4), i.e. theoretically each module can be replaced with one having the same interface, but different behavior, without affecting architecture stability. Due to (R.1), $\mathcal{H}$TS must adopt an event-based architecture (R.5) as normally done by RSP engines and it must exploit a simple to parse RDF serialization for events (R.6) to minimize the cost of putting an existing RSP engine on the test stand.

[2] https://github.com/streamreasoning/heaven

A minimal measurements set for a relevant RSP engine evaluation is defined in [15, 7]. $\mathcal{H}$TS must include it, i.e (R.7a) *Latency* – the delay between the event injection into the RSP engine and the system response; (R.7b) *Memory Usage* – the difference between total system memory and free memory; (R.7c) *Completeness & Soundness* of query-answering results w.r.t the system entailment regime [7]. To allow further development, the performance measurement set that $\mathcal{H}$TS can collect must be also extensible (R.8).

The RSP engines input-output relationship is non-trivial [7], due to their I/O asynchronous nature. Indeed, experiments *Repeatability* requires $\mathcal{H}$TS to control the experiment execution without affecting the RSP engine evaluation. To this extent, $\mathcal{H}$TS must not be running concurrently with the RSP engine (R.9) and $\mathcal{H}$TS must have a reduced (and possibly constant) memory footprint (R.10) to do not affect the reasoning performances.

Finally, experiments *Comparability* requires $\mathcal{H}$TS to support the collection of the performance measurements for post-hoc analysis (R.11).

3 $\mathcal{H}$eaven Architecture & Workflow

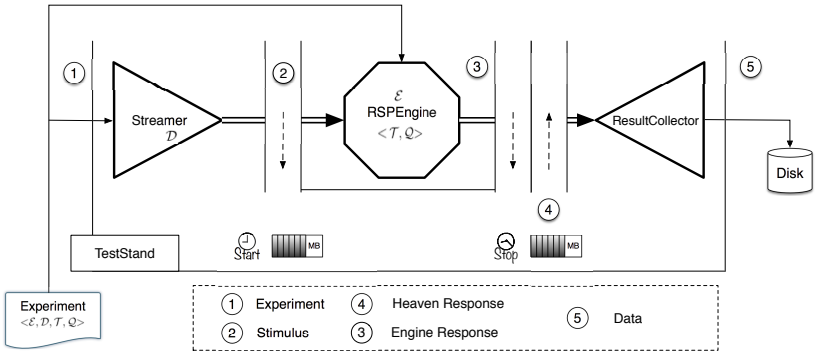

Fig. 1: $\mathcal{H}$eaven modules and workflow

In this section we describe $\mathcal{H}$TS architecture and also indicate which requirements are satisfied at architectural level.

The tuple $< \mathcal{E}, \mathcal{D}, \mathcal{T}, \mathcal{Q} >$ describes the top-level input of $\mathcal{H}$TS: an Experiment. $\mathcal{E}$ is the evaluated RSP engine (satisfying R.1); $\mathcal{D}$ is the description of the incoming data flow (satisfying R.2a); $\mathcal{T}$ is the ontology (satisfying R.2b); $\mathcal{Q}$ the continuous queries set registered into $\mathcal{E}$ (satisfying R.3).

In Step (1) $\mathcal{H}$TS receives such an Experiment and it independently initializes each module: the engine $\mathcal{E}$ needs to be initialized by registering into the ontology $\mathcal{T}$ and all the queries in $\mathcal{Q}$; the Streamer, which is the actual data stream source, uses the description $\mathcal{D}$ to build the incoming information flow (e.g. an RDF Streams) to push into $\mathcal{E}$; the Result Collector starts listening for the results of $\mathcal{E}$, that it will persist for post-hoc analysis (satisfying R.11).

During experiment execution, $\mathcal{H}$TS loops through the steps from (2) to (5) until the ending condition is reached (e.g the end of the dataset). It exchanges three kinds of events: (i) Stimulus (S) a portion of the input information flows in which all triples have the same timestamps; (ii) Engine Response ($\mathcal{E}R$) the event format that $\mathcal{E}$ is required to output. It contains the answer to one of the query in the query-set $\mathcal{Q}$ registered in $\mathcal{E}$ given the ontology $\mathcal{T}$ and the active window in the RSP engine; (iii) $\mathcal{H}$eaven Response ($\mathcal{H}R$) which encapsulates the $\mathcal{E}R$ content adding the performance measurements collected by $\mathcal{H}$TS.

In step (2), the Streamer builds and pushes to $\mathcal{E}$ an event S. Before starting step (3), $\mathcal{H}$TS starts a timer to measure latency (satisfying R.4a) and it measures the memory load (satisfying R.4b).

In step (3), $\mathcal{H}$TS invokes the engine $\mathcal{E}$ and then it waits until $\mathcal{E}$ completes its processing. We can be sure that $\mathcal{H}$TS and the engine are concurrently executing, because $\mathcal{H}$TS is a finite state machine. $\mathcal{H}$TS stops the timer and measures again the memory load as soon as $\mathcal{E}$ returns it the control.

In step (4), $\mathcal{H}$TS creates a $\mathcal{H}R$ adding to the produced $\mathcal{E}R$ the collected performance measurements. $\mathcal{H}$TS pushes the $\mathcal{H}R$ to the Result Collector.

In step (5), the Result Collector persists $\mathcal{H}R$ content (satisfying R.11).

4 Experiment Design

In this section we explain how to design each element of the tuple $< \mathcal{E}, \mathcal{D}, \mathcal{T}, \mathcal{Q} >$ according to the current $\mathcal{H}$TS implementation. We also motivate the development choices we did, by the means of the proposed requirements (Section 2).

The $\mathcal{E}$ specification is realized by the means of a facade pattern [8, pp. 243]. An abstract RSP engine class allows to associate any RSP engines ensuring engine independence (satisfying R.1), it intercepts the S and taking care of the $\mathcal{E}R$ creation.

To simplify the initial usage of $\mathcal{H}$TS, the current implementation contains four naïve RSP engine implementations with external time-control, that we be natively used as $\mathcal{E}$. Some results about their performances, evaluated using $\mathcal{H}$TS current implementation are already available[3].

$\mathcal{D}$ comprises all the details of the incoming input flows: (i) the actual data to stream (e.g. a dataset); (ii) the number of the incoming data streams, (iii) their data model (e.g. RDF Stream); (iv) their flow rate profile (e.g. Gaussian, Poisson) and an ending condition for the experiment (e.g. max number of events). The Streamer interface allows to describe all these details according with the user needs. The current $\mathcal{H}$TS release comprises the RDF2RDFStream, that adapts any RDF dataset to a streaming scenario (RDF Stream) by adding to a Stimulus the required number of triples to build such an event. Different flow profiles can be realized by the means of the FlowRateProfiler. In our testing experiments, we adapted one generation of LUBM(1000, 0)[4] to our purpose.

[3] http://streamreasoning.org/TR/2015/Heaven/iswc2015-appendix.pdf

[4] LUBM(N, S) indicates the dataset with N universities generated using a seed S.

Moreover, RDF triples in S and $\mathcal{E}R$ are encoded in the N-Triple format[5], an easy-to-parse RDF serialization used by the majority of exiting RSP engines (satisfying R.6).

The ontology $\mathcal{T}$ is chosen according with the background knowledge required for the reasoning. During the execution of an experiment, $\mathcal{T}$ is considered to be static. Indeed is a good practice to reduce the reasoning complexity by executing its materialization at setup-time. In the current implementation we used the RDFS version of the LUBM ontology as $\mathcal{T}$.

The query set $\mathcal{Q}$ must be register directly to $\mathcal{E}$, because $\mathcal{H}$TS does not offer API yet. For our evaluation, we used a single query for each engine for each experiment. They were variants of the full the materialization under ρDF [13] entailment regime[6]. The query results are appended to an open file in order to minimize the memory usage (satisfying R.10).

Notably, $\mathcal{H}$TS is currently implemented as a single thread application (satisfying R.9). Completeness and Soundness (C&S) of the query results and the $\mathcal{H}$TS ability to know exactly what was sent into $\mathcal{E}$ are evaluated post-hoc (R.7c) by using the content of the persisted $\mathcal{H}R$ events. Real time verification of C&S is also possible for those engines which allow external time control, because $\mathcal{H}$TS satisfies (R.9), but it may violate requirement (R.10), due to the large memory footprint of reasoning procedures.

5 Conclusions

In this paper we presented the requirements and the architecture of $\mathcal{H}$eaven Test Stand, an infrastructure to enable a Systematic Comparative Research Approach for RSP engines, by the means of experiment design and execution. A further contribution is a working implementation of $\mathcal{H}$TS, already available on GitHub[7].

Enabling SCRA is a crucial step towards an efficient and effective stream reasoning, but, as first future work, we have to prove $\mathcal{H}$TS usability and effectiveness of the proposed implementation. Solving the former requires us to go step by step to the experiment design and execution; the latter, instead, requires to provide experimental proofs that $\mathcal{H}$TS influences the engine performance in a controlled and predictable way only and it does not affect the query results.

Another future work consists into better comprehend the state-of-the-art solution space, that means benchmarking mature solutions like the C-SPARQL [1] engine or CQELS [10].

We want to make available the performance analysis, together with the experiments setting we use for their evaluation (i.e. the $< \mathcal{E}, \mathcal{D}, \mathcal{T}, \mathcal{Q} >$ configuration). This is clearly necessary, because we want $\mathcal{H}$TS to be adopted by the entire SR

[5] http://www.w3.org/2001/sw/RDFCore/ntriples/

[6] All the queries have the same sliding parameter $\beta = 100$ ms and they differ for the duration ω. In particular, we use time-based sliding windows in which ω is an integer multiple of the slide parameter β, i.e., it holds that $\omega = \beta * S$ where S is a positive integer.

[7] https://github.com/streamreasoning/heaven

community. Thus, another pioneering step consists into developing a methodology for the result analysis. Such methodology should exploits statistical techniques to interpret the obtained results and allows us to finally answer questions like *Qualitatively, which is the best solution?* or *Quantitatively, what distinguish a solution from other ones?* or again *Why solution A performs better than B under a certain experimental condition?*

Acknowledgments. This work has been partially funded by the IBM faculty award 2013 granted to prof. Emanuele Della Valle and by the IBM PhD fellowship award granted to Daniele Dell'Aglio.

References

1. Barbieri, D.F., Braga, D., Ceri, S., Della Valle, E., Grossniklaus, M.: C-SPARQL: A continuous query language for RDF data streams. IJSC 4(1), 3–25 (2010)
2. Botan, I., Derakhshan, R., Dindar, N., Haas, L., Miller, R.J., Tatbul, N.: Secret: A model for analysis of the execution semantics of stream processing systems. PVLDB 3(1), 232–243 (2010)
3. Calbimonte, J.P., Corcho, O., Gray, A.J.G.: Enabling ontology-based access to streaming data sources. In: ISWC. pp. 96–111 (2010)
4. Creswell, J.W.: Research Design: Qualitative, Quantitative, and Mixed Methods Approaches. Sage Publications Ltd., 3 edn. (2008)
5. Cugola, G., Margara, A.: Processing flows of information: From data stream to complex event processing. ACM Comput. Surv. 44(3), 15:1–15:62 (Jun 2012)
6. Della Valle, E., Ceri, S., van Harmelen, F., Fensel, D.: It's a streaming world! reasoning upon rapidly changing information. IEEE Intelligent Systems 24(6), 83–89 (2009)
7. Dell'Aglio, D., Calbimonte, J., Balduini, M., Corcho, Ó., Della Valle, E.: On correctness in RDF stream processor benchmarking. In: ISWC. pp. 326–342 (2013)
8. Freeman, E., Freeman, E., Bates, B., Sierra, K.: Head First Design Patterns. O' Reilly & Associates, Inc. (2004)
9. Komazec, S., Cerri, D., Fensel, D.: Sparkwave: continuous schema-enhanced pattern matching over RDF data streams. In: DEBS. pp. 58–68 (2012)
10. Le-Phuoc, D., Dao-Tran, M., Xavier Parreira, J., Hauswirth, M.: A native and adaptive approach for unified processing of linked streams and linked data. In: ISWC. pp. 370–388 (2011)
11. Le-Phuoc, D., Dao-Tran, M., Pham, M.D., Boncz, P., Eiter, T., Fink, M.: Linked stream data processing engines: Facts and figures. In: ISWC. pp. 300–312 (2012)
12. Margara, A., Urbani, J., van Harmelen, F., Bal, H.E.: Streaming the web: Reasoning over dynamic data. J. Web Sem. 25, 24–44 (2014)
13. Muñoz, S., Pérez, J., Gutiérrez, C.: Minimal deductive systems for RDF. In: ESWC. pp. 53–67 (2007)
14. Rihoux, B., Ragin, C.C.: Configurational comparative methods: Qualitative comparative analysis (QCA) and related techniques. Sage (2009)
15. Scharrenbach, T., Urbani, J., Margara, A., Della Valle, E., Bernstein, A.: Seven commandments for benchmarking semantic flow processing systems. In: ESWC. pp. 305–319 (2013)
16. Zhang, Y., Duc, P., Corcho, O., Calbimonte, J.P.: SRBench: A Streaming RDF/SPARQL Benchmark. In: ISWC. pp. 641–657 (2012)