Stressless RSP Benchmarking With RSPLab

Andrea Mauri, Riccardo Tommasini, Emanuele Della Valle, Marco Brambilla

Politecnico di Milano, DEIB, Milan, Italy
`{name.lastname}@polimi.it`

Abstract. RSPLab is an infrastructure that reduces the effort required to design and execute reproducible experiments as well as share their results. It provides a programmatic environment to deploy RDF Streams and RSP engines, interact with them and continuously monitor their performances and collect statistics. In this demo well show how to deploy RDFStream and RSP engines in the cloud and how to interact with them registering and unregistering continuous queries lively. We will show how these action impact the performances and we transform the results in visual plot on a chart.

Keywords: Semantic Web, Stream Reasoning, RDF Stream Processing, Benchmarking

1 Introduction

In the RDF Stream Processing (RSP) community particular interest has arisen on empirical research on benchmarks and evaluation methodologies. These comprehend studies on query language expressive power, performance, correctness of results, memory load and latency [3] (§ 2.4).

Beside community efforts, the evaluation of RSP engines is still not systematic. This mostly because RSP engines have different semantics, interfaces, and processing models. Moreover, existing test drivers [5, 1] are benchmark specific. This situation challenges comparability of the experiments, because it forces researchers to set up ad-hoc configuration environments that might have architectural bias and produce an unmeasurable experimental errors.

In this paper, we present RSPLab [7] a cloud-ready open-source test driver to support empirical research for RSP.

We will show how it enables design of experiments by the means of a programmatic interface that allows deploying the environment, running experiments, measuring the performance, visualizing the results as reports, and cleaning up the environment to get ready for a new experiment.

The remainder of the paper is structured as follow: Section 2 shows the RSPLab architecture, Section 3 describes how the demonstration will be carried out and finally Section 4 concludes and provides information on where additional material regarding the resource can be found.

2 RSPLab

Figure 1 presents RSPLab architecture that comprises four independent tiers: *Streamer*, *Consumer*, *Collector* and *Controller*.

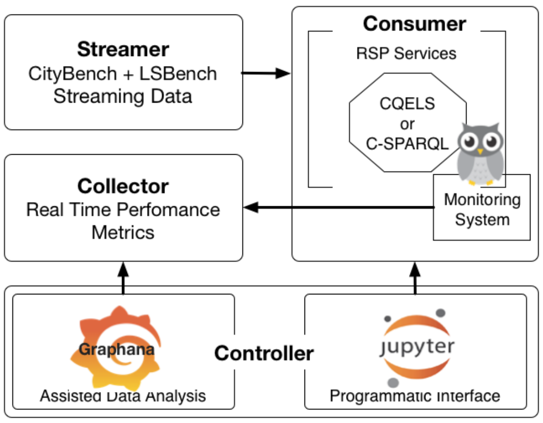

Fig. 1: RSPLab Architecture

RSPLab was developed using Docker, i.e. a lightweight virtualization framework[1]. This allows to fully control the available resources enabling experiments execution at scale and it guarantees components isolation. Moreover, it fosters reproducibility by making the execution hardware-independent.

Streamer. The data provisioning tier publishes RDF streams from existing sources. It is implemented using a modified version of TripleWave [4][2]

Consumer. The data processing tier exposes the RSP engines on the web using the RSP Services [2], generalizing the processing model enabling streams registration, queries registration and results consumption.

Collector. This tiers is in charge of monitoring the execution of a benchmark, in particular it includes: (1) a distributed continuous monitoring system[3], that collects performance statistics, (2) a time-series database[4] which is used to store the data collected and (3) a RSPSink that persists query results on a cloud file systems, allowing for post-hoc correctness analyses.

Controller. The control and analysis tier allows the RSPLab user to interact with the whole environment by deploying streams and engines and monitoring the performance of the latter. A programmatic interface, implemented using iPython Notebooks[5] includes wrappers for RSP services, TripleWave and RSPsinks. Assisted data visualization dashboards are accessible, in real time, via Grafana[6], i.e. a dashboard that reads directly form the time-series database. Experimental reports annotated with VOID can be generated also using the library.

1 `https://www.docker.com/`
2 `https://github.com/streamreasoning/triplewave/tree/rsplab`
3 `https://github.com/google/cadvisor`
4 `https://www.influxdata.com/`
5 `https://ipython.org/notebook.html`
6 `https://grafana.com/`

3 Demonstration

```
_Q1 = rsp.BenchmarkQueries.CityBench.Q1
e = rsp.new_experiment()
e.add_engine("http://cqles.rsp-lab.org", 80, rsp.Dialects.CQELS)
e.add_KPIs(rsp.KPI.Memory_Consumption, rsp.KPI.CPU_Load)
e.add_query("CB.Q1", rsp.QueryType.Query, _Q1, rsp.Dialects.CQELS)
e.add_tbox("CB.Q1", name="citytraffic.owl", base="rsp-lab.org")
e.add_graph("CB.Q1", name="SensorRepository.rdf", base="rsp-lab.org")
e.add_stream("CB.Q1","AarhusTrafficData158505", base="rsp-lab.org")
e.add_stream("CB.Q1","AarhusTrafficData182955", base="rsp-lab.org")
```

Listing 1.1: Experiment Design

```
#WARM-UP

rsp = RSPEngine(ehost, eport);

for d in experiment.graphs():
 rsp.register_graph(d)
for s in experiment.streams():
 rsp.register_stream(s)
for q in experiment.queries():
 rsp.register_query(q)
 rsp.new_observer(query, 'default')
spawn_sinks(experiment)

# OBSERVE
wait(experiment.duration())

for q in engine.queries():
  for o in engine.observers(q):
     rsp.unregister_observer(o)
  rsp.unregister_query(q)
for s in engine.streams():
  rsp.unregister_stream(s)
rsp.report.publish(experiment)
```

Listing 1.2: Simple execution logic

In this demo we show all the workflow necessary for deploying and executing a RSP benchmark using RSPLab. In particular we will present the following steps:

Step-1 Deployment and configuration of RDF Streams: we show how to deploy and configure the *Streamer* component in order to stream the included datasets.
Step-2 Deployment and configuration of RSP Engines: we show how to deploy and configure the *Consumer tier*, with particular focus on the C-SPARQL and CQELS engines.
Step-3 Continuous query of the streams and collection of the results: we show how to issue continuous queries to the streams using the programming interface. We use the included Python library[7] that allow to fully control all the tiers included in RSPLab. In particular it allows to (1) create and launch experiments following the structure described in [6], (2) dynamically add and remove graphs and streams to a RSP engine, (3) dynamically register and unregister queries and observers to a RSP engine, and finally (4) publish the results following the linked data principles. Listing 1.1 and 1.2 show respectively the basic code necessary to create an experiment and and to implement a simple logic for executing a benchmark. Notice that a RSPLab user is free to build any execution workflow according to his requirements.
Step-4 Visualization of statistics in real time: we show how to use Grafana on the *Controller* tier in order to visualize the performance indexes in real-time (an example is shown in Figure 2). Moreover we will show how with RSPLab is possible to observe the different phases of an experiment: the warm-up phase, where the engines are deployed and the streams and the queries are registered, and the observation phase, where the engines are in steady state, consuming the streams and answering the queries.

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

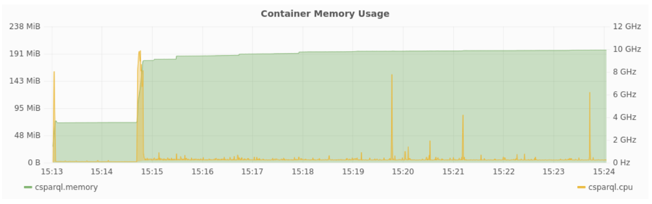

Fig. 2: A example of dashboard showing CPU load and Memory usage.

With this demonstration we aim to show the effectiveness of RSPLab in facilitating the deployment and execution of RSP benchmarks. We will run both the included benchmarks (e.g., Citybench and LSBench) and custom variation, by adding and removing queries in real-time, in order to highlight how these actions impact on the performance.

4 Conclusion

The main contribution of this demonstration paper is the proof of RSPLab effectiveness. The Docker-based architecture and the programmatic interface facilitate the process of deploying the environment, running experiments, measuring the performance, visualizing the results as reports.

RSPLab is released as open-source[8], examples, documentation and tutorials are available on GitHub[9], hosted by the Stream Reasoning organization.

References

1. Ali, M.I., Gao, F., Mileo, A.: Citybench: A configurable benchmark to evaluate RSP engines using smart city datasets. In: The Semantic Web - 14th ISWC, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part II. pp. 374–389
2. Balduini, M., Valle, E.D.: A restful interface for RDF stream processors. In: Proceedings of the ISWC 2013 Posters & Demonstrations Track, Sydney, Australia, October 23, 2013. pp. 209–212
3. DellAglio, D., Della Valle, E., van Harmelen, F., Bernstein, A.: Stream reasoning: A survey and outlook. Data Science (Preprint), 1–24
4. Mauri, A., Calbimonte, J., Dell'Aglio, D., Balduini, M., Brambilla, M., Della Valle, E., Aberer, K.: Triplewave: Spreading RDF streams on the web. In: The Semantic Web - 15th International Semantic Web Conference, Kobe, Japan, October 17-21, 2016, Proceedings, Part II. pp. 140–149
5. Phuoc, D.L., Dao-Tran, M., Pham, M., Boncz, P.A., Eiter, T., Fink, M.: Linked stream data processing engines: Facts and figures. In: The Semantic Web - 11th ISWC, Boston, MA, USA, November 11-15, 2012, Proceedings, Part II. pp. 300–312
6. Tommasini, R., Della Valle, E., Balduini, M., Dell'Aglio, D.: Heaven: a framework for systematic comparative research approach for rsp engines. In: 13th Extended Semantic Web Conference, ESWC 2016, Heraklion, Crete, Greece. pp. 87–92
7. Tommasini, R., Valle, E.D., Mauri, A., Brambilla, M.: Rsplab: Rdf stream processing benchmarking made easy. In: The Semantic Web 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part II

[8] `http://rsp-lab.org`
[9] `https://github.com/streamreasoning/rsplab`