PRSP: A Plugin-based Framework for RDF Stream Processing

Qiong Li
School of Computer Science and Technology,Tianjin University
Tianjin Key Laboratory of Cognitive Computing and Application
Tianjin 300350, P.R. China
liqiong@tju.edu.cn

Xiaowang Zhang*
School of Computer Science and Technology,Tianjin University
Tianjin Key Laboratory of Cognitive Computing and Application
Tianjin 300350, P.R. China
*Corresponding author
xiaowangzhang@tju.edu.cn

Zhiyong Feng
School of Computer Software,Tianjin University
Tianjin Key Laboratory of Cognitive Computing and Application
Tianjin 300350, P.R. China
zyfeng@tju.edu.cn

ABSTRACT

In this paper, we propose a plugin-based framework for RDF stream processing (PRSP). With this framework, we can apply SPARQL engines to process C-SPARQL queries with maintaining the high performance of those engines in a simple way. Taking advantage of PRSP, we can process large RDF streams in a distributed context via distributed SPARQL engines. Moreover, we can evaluate the performance and correctness of existing SPARQL engines in handling RDF streams in a united way, which amends the evaluation of them ranging from static RDF to dynamic RDF. Finally, we experimentally evaluate the performance and correctness of PRSP on YABench. The experiments show that PRSP can still maintain the high performance with SPARQL engines in RDF stream processing although there are some slight differences among them.

Keywords

RDF Stream; RSP; C-SPARQL; SPARQL

1. INTRODUCTION

RDF stream, as a new type of dataset, can model real-time and continuous information in a wide range of applications, e.g., environmental monitoring, smart cities and so on. But data stream is unbounded sequences of time-varying data element and difficult to store. What is more, there are few RSP[1] (RDF stream processing) systems, such as C-SPARQL[2], implemented for supporting RDF streams due to its complicacy in processing. On the other hand, there are many popular and efficient SPARQL engines supporting only static RDF graphs, such as centralized engines, Jena, RDF-3X and gStore, and distributed systems, TriAD[3], gStoreD [5] and so on. So, it is an interesting problem to evaluate continuous queries by employing SPARQL engines.

In this paper, we provide a plugin-based framework for RDF stream processing named PRSP, which makes it possible to use the high-performance SPARQL engines valid only for RDF graphs, to process RDF streams. Moreover, within this framework, we can exploit any SPARQL engines to process RDF streams in a convenient way and compare their performance under a unified framework. And users can choose the favourable systems based on their all kinds of requirements. For example, they have the need to handle large-volume RDF streams, thus distributed engines are the best choice.

WWW 2017 Companion, April 3–7, 2017, Perth, Australia.
ACM 978-1-4503-4914-7/17/04.
http://dx.doi.org/10.1145/3041021.3054243

2. C-SPARQL TO SPARQL

RDF stream An RDF stream S is defined as ordered sequences of pairs, made of an RDF triple and a timestamp τ: $(\langle s,p,o\rangle,\tau)$.
C-SPARQL query Formally, a C-SPARQL query Q can be taken as a 5-tuple of the form $Q = [\mathrm{Req}, S, \mathrm{w}, \mathrm{s}, Q_{\mathrm{SPARQL}}]$, where

- Req: the registration;
- S: the RDF stream registered;
- w: RANGE, i.e., the window size;
- s: STEP, i.e., the updating time of windows;
- Q_{SPARQL}: a SPARQL query.

Let Q be a C-SPARQL query, we use $\mathrm{Req}(Q)$, $S(Q)$, $\mathrm{w}(Q)$, $\mathrm{s}(Q)$, and $Q_{\mathrm{SPARQL}}(Q)$ to denote the registration, the RDF stream registered, RANGE, STEP, and the SPARQL query of Q.

Now, we consider an example: Let Q_{Temp} be a C-SPARQL query *TempQuery* shown as follows:

```
1. REGISTER QUERY TempQuery AS
2. SELECT ?obs
3. FROM STREAM TempStream [RANGE 5s   STEP 4s ]
4. WHERE { ?obs observedProperty AirTemperature . }
```

Based on our proposed formalization, we can find:

- $\mathrm{Req}(Q_{\mathrm{Temp}}) =$ **REGISTER QUERY** *TempQuery* AS;
- $S(Q_{\mathrm{Temp}}) =$ *TempStream*;
- $\mathrm{w}(Q_{\mathrm{Temp}}) = 5s$;
- $\mathrm{s}(Q_{\mathrm{Temp}}) = 4s$;
- $Q_{\mathrm{SPARQL}}(Q_{\mathrm{Temp}})$ is a SPARQL query as follows:

```
1. SELECT ?obs
2. WHERE { ?obs observedProperty AirTemperature . }
```

A (logical) *window* denoted by $G(S, \mathrm{w}, \mathrm{s}, t)$ for an RDF stream S, a window size (RANGE) w, an updating time of windows (STEP) s, and a time t (as the present time) is a collection of triples defined as: where $k = 0, 1, 2, \ldots$ and $t' = t + k\mathrm{s}$,

$$G_k(S, \mathrm{w}, \mathrm{s}, t) = \{\langle s,p,o\rangle \mid (\langle s,p,o\rangle, \tau) \in S \text{ and } t' - \mathrm{w} \leq \tau \leq t'\}.$$

Now, we can conclude the main result of this poster:

THEOREM 1. *Let Q be a C-SPARQL query. For any RDF stream S and any present time t, if S is registered in Q then for any $k = 0, 1, 2, \ldots$, we have $[\![Q]\!]_{[S,t]} = [\![Q_{\mathrm{SPARQL}}]\!]_{G_k(S,\mathrm{w},\mathrm{s},t)}$.*

3. FRAMEWORK OF PRSP

PRSP is an extension of SPARQL for querying over RDF streams shown in Figure 1. Both C-SPARQL query and RDF streams as the input of PRSP are transformed by the plugin query rewriting and data transformer respectively. After that, the output from the former plugins as the input of SPARQL API, the results are obtained by a SPARQL query engine. And the right box, consisting of any SPARQL engine, is used as a black box to evaluate RDF graphs.

Query Rewriting A C-SPARQL query Q as the input of query rewriting mode, generate SPARQL query (i.e., Q_{SPARQL}) and window selector (i.e., $\mathrm{w}(Q)$ and $\mathrm{s}(Q)$), which can be addressed in SPARQL API and data transformer module respectively.

Data Transformer The data transformer module handles RDF streams via a DSMS. And it transforms RDF streams into RDF graphs $G_k(S, \mathrm{w}, \mathrm{s}, t)$ based on the window size and step set at window operator.

SPARQL API Our proposed plugin-based framework for RDF stream processing (PRSP) introduces a unified interface for RDF engines, in which a trigger (serving as Req, i.e., the registration of C-SPARQL query) for pushing SPARQL queries periodically and SPARQL API for running SPARQL engines.

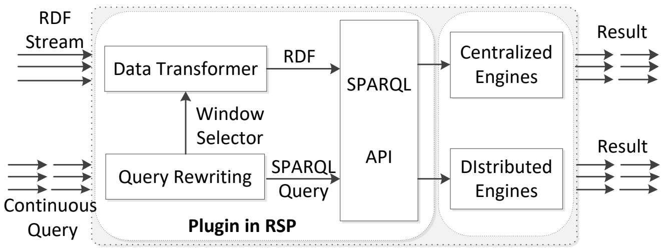


Figure 1: The framework of PRSP

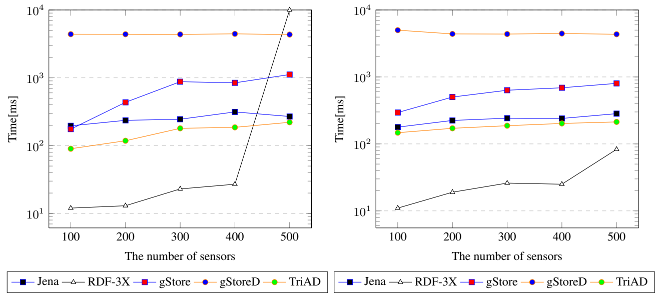
4. EXPERIMENTS AND EVALUATIONS

All centralized experiments were carried out on a machine running Linux, which has 4 CPUs with 6 cores and 64GB memory, and 5 nodes with the same performance for distributed experiments. We utilized YABench RSP benchmark [4], which uses a real world dataset about water temperature. In our experiments, we performed tumbling windows with a window size and step of 5 seconds respectively, and chose two BGP queries, Q_1 and Q_2. Q_1 is a BGP query with four forms from YABench, and Q_2 is the rewriting of Q_1 with three triples.

Table 1: Precision/Recall results

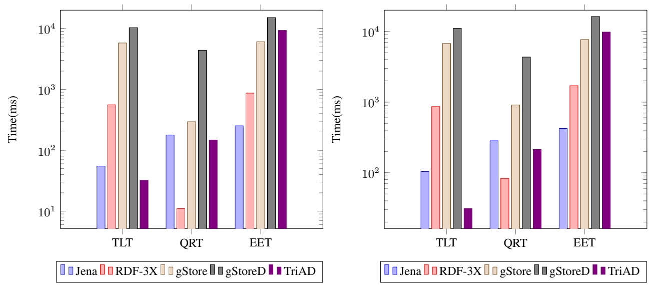
	sensor	Jena	RDF-3X	gStore	gStoreD	TriAD
Precision	100	99%	93%	100%	100%	97%
	300	97%	94%	100%	100%	93%
	500	85%	88%	100%	100%	100%
Recall	100	95%	89%	75%	72%	95%
	300	94%	91%	88%	76%	92%
	500	92%	79%	77%	63%	91%

The performance of each engine is shown in Fig 2, which is carried out under the five different input loads for windows using the two queries, Q_1 (Fig 2(a)) and Q_2 (Fig 2(b)). When the load ranges



(a) The response time of Q_1 (b) The response time of Q_2

Figure 2: Response time in different scenarios within PRSP



(a) s = 100 sensors (b) s = 500 sensors

Figure 3: RDF stream for processing time in PRSP

from $s = 100$ sensors to $s = 500$ sensors, the query response time is increased in varying degrees except for gStoreD [5]. Fig 3 shows the time of three processes under $s = 100$ (Fig 3(a)) and $s = 500$ (Fig 3(b)) obtained from query Q_2, including triples load time (*TLT*), query response time(*QRT*), and engine execution time (*EET*). *TLT* from RDF-3X, gStore, and gStoreD except TriAD occupies a large part of *EET*, resulting in their lower efficiency for processing RDF streams. Table 1 illustrates the results of precision and recall from the experiments under three load scenarios (i.e., $s = 100/300/500$) in PRSP. Along with more input load for windows, most of them enjoy lower recalls with high accuracy.

5. ACKNOWLEDGMENTS

This work is supported by the program of the National Natural Science Foundation of China (61672377), the National Key Research and Development Program of China (2016YFB1000603), and the Key Technology Research and Development Program of Tianjin (16YFZCGX00210).

6. REFERENCES

[1] `http://www.w3.org/community/rsp/`.
[2] D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, and M. Grossniklaus. Querying RDF streams with C-SPARQL. *ACM SIGMOD Record*, 39(1):20–26, 2010.
[3] S. Gurajada, S. Seufert, I. Miliaraki, and M. Theobald. TriAD: a distributed shared-nothing RDF engine based on asynchronous message passing. In: *Proc. of SIGMOD'14*, pp.289–300, 2014.
[4] M. Kolchin, P. Wetz, E. Kiesling, and A. M. Tjoa. YABench: A comprehensive framework for RDF stream processor correctness and performance assessment. In: *Proc. of ICWE'16*, pages 280–298. Springer, 2016.
[5] P. Peng, L. Zou, M. T. Özsu, L. Chen, and D. Zhao. Processing sparql queries over distributed rdf graphs. *VLDB J.*, 25(2):243–268, 2016.