Web Semantics: Science, Services and Agents on the World Wide Web 42 (2017) 38–54

Contents lists available at ScienceDirect

Web Semantics: Science, Services and Agents on the World Wide Web

journal homepage: www.elsevier.com/locate/websem

Operator-aware approach for boosting performance in RDF stream processing

Danh Le-Phuoc

Insight Centre for Data Analytics, National University of Ireland, Galway, Ireland

ARTICLE INFO

Article history:
Received 16 July 2015
Received in revised form
7 March 2016
Accepted 1 April 2016
Available online 11 April 2016

Keywords:
Continuous queries
Linked stream data
Linked data
Semantic web
Stream processing

ABSTRACT

To enable efficiency in stream processing, the evaluation of a query is usually performed over bounded parts of (potentially) unbounded streams, i.e., processing windows "slide" over the streams. To avoid inefficient re-evaluations of already evaluated parts of a stream in respect to a query, incremental evaluation strategies are applied, i.e., the query results are obtained incrementally from the result set of the preceding processing state without having to re-evaluate all input buffers. This method is highly efficient but it comes at the cost of having to maintain processing state, which is not trivial, and may defeat performance advantages of the incremental evaluation strategy. In the context of RDF streams the problem is further aggravated by the hard-to-predict evolution of the structure of RDF graphs over time and the application of sub-optimal implementation approaches, e.g., using relational technologies for storing data and processing states which incur significant performance drawbacks for graph-based query patterns. To address these performance problems, this paper proposes a set of novel operator-aware data structures coupled with incremental evaluation algorithms which outperform the counterparts of relational stream processing systems. This claim is demonstrated through extensive experimental results on both simulated and real datasets.

1. Introduction

There are billions of heterogeneous stream data sources that are continuously producing an enormous amount of information under diverse ownerships and controls. To overcome this issue, the RDF data model becomes a natural choice to provide an integrated view for querying the data without requiring an adherence to a specified schema. Hence, several RDF Stream Processing (RSP) engines [1] have been proving their advantages in tackling interoperability in several projects and products in Internet of Things and Smart Cities, etc. However, due to the intrinsic nature of stream data processing in such targeted applications, the ability to process stream data at high throughput and low latency is a vital requirement that has not been thoroughly addressed in such engines.

A common strategy in implementing an RSP so far is delegating performance issues to underlying libraries or systems, e.g., RDF/SPARQL query processors, data stream management systems (DSMSs) or complex event processing engines (CEPs). This strategy greatly benefits from the available techniques, approaches and tools provided by such research communities. However we believe that it requires a lot of effort in investigation, tuning and re-engineering to build a performant RSP engine. In particular, the RDF store or SPARQL query processor is designed for heavily read-intensive contexts [2–4] whilst an RSP engine needs to deal with high writing throughput of unbounded incoming data and continuous computation corresponding to such updates. To answer this need, several RSP engines employ the work on evaluating sliding-window operators of DSMS or CEP. For instance, C-SPARQL [5] uses ESPER[1] together with Jena[2] and CQELS [6] implements operators of Aurora [7] using the underlying TDB libraries of Jena.

In such generic stream-processing engines, there are two evaluation strategies of executing sliding-window operators, i.e., re-evaluation and incremental evaluation [8]. In the re-evaluation strategy, the query is re-evaluated independent of the previous computation efforts. To avoid inefficient re-evaluations of already-evaluated parts of a stream in respect to a query, incremental-evaluation strategies are applied, i.e., the query results are obtained incrementally from the result set of the preceding processing state

E-mail address: danh@danhlephuoc.info.

http://dx.doi.org/10.1016/j.websem.2016.04.001
1570-8268/

[1] http://www.espertech.com/esper/index.php.
[2] https://jena.apache.org/.

without having to re-evaluate all input buffers. This method has been proven highly efficient to some extent [9–13] but comes at the cost of having to maintain processing state, which is not trivial, and may defeat performance and scalability advantages of the incremental evaluation strategy [11,8]. As shown in [8], several efforts in providing incremental evaluation algorithms for sliding windows have been conducted over the years but there are still challenging issues and a lot of room for improvement. In the context of RDF streams, the problem is further aggravated by the hard-to-predict evolution of the structure of RDF graphs over time and the application of sub-optimal implementation approaches, e.g., using relational technologies for storing data and processing states which incur significant performance drawbacks for graph-based query patterns.

To address the above shortcomings and challenging issues, this paper introduces a set of novel operator-aware data structures associated with efficient incremental evaluation algorithms to deal with the specific properties of RDF stream data and common query patterns. These new data structures are designed to handle small data items and intermediate processing states very efficiently. The data structures include various low-maintenance-cost indexes to support high throughput in the probing operations that are used in various operator implementations. Based on such data structures, we propose several algorithms to enable incremental evaluation of basic operators such as join, aggregation, duplicate elimination and negation. These algorithms also overcome the typical problems of incremental evaluation of windowing operators. Our experimental results show that our approach performs better than relational approaches and re-evaluation approaches by orders of magnitude in reducing query execution delay.

The remainder of this article is structured as follows. In Section 2 we introduce some background concepts and techniques and present some analyses of current approaches to set the context of the contributions of the paper. Then, we describe our operator-aware data structures in Section 3, including storage design, access methods, usage, implementation variation, optimisation, and performance tuning. After that we present incremental evaluation algorithms based on these data structures in Section 4. To evaluate the performance of the operators built on such data structures and algorithms, we present and discuss the experimental results using our new data structures and algorithms in Section 5. Besides, the related work is also discussed in Section 6. Finally, we finish the paper with our conclusions.

2. Background and analysis

Before going into technical details of the article, this section introduces some background to review the technical shortcomings that motivate the design of our data structures and algorithms.

2.1. Continuous query operators on RDF stream

An RDF stream is modelled by utilising the definitions of RDF nodes and RDF triples whereby the *stream elements* of an RDF stream are represented as RDF triples with temporal contexts. As the standardisation of RDF Streams is still the on-going work of the W3C RSP community,[3] this paper will leave the formal definitions of an RDF stream generic enough for latter adoption. Instead, we focus on evaluation aspects of basic continuous query operators, e.g., join, aggregation and duplication elimination. Hence, we introduce an example of real RDF streams to investigate the issues of evaluating continuous queries inspired by the open dataset of taxi rides in New York.[4] Note that instead of following strictly to its stream format, we assume that there are three RDF streams, $\mathcal{S}_{pickup}$, $\mathcal{S}_{dropoff}$ and $\mathcal{S}_{fare}$ with the schema shown below. Two streams $\mathcal{S}_{pickup}$ and $\mathcal{S}_{dropoff}$ report the events of a taxi being picked up or dropped off respectively. The stream $\mathcal{S}_{fare}$ reports the payment for a taxi ride that was reported in the stream $\mathcal{S}_{pickup}$ at a picking-up time given by the triple with the predicate :*pickupTime*. Note that in the example, an event is represented as an RDF graph, however, clearly other representations still can be processed by the operators to be discussed in this paper.

$$\mathcal{S}_{pickup} = \left\{ \begin{array}{l} :ride_1 \; :taxi \;\; : 89...CF4 \\ :ride_1 \; :pickupTime \; "2013-01-0115:11:48". \end{array} \right\}.$$

$$\mathcal{S}_{dropoff} = \left\{ \begin{array}{l} :ride_1 \; :dropoffTime \; "2013-01-0115:18:10". \\ :ride_1 \; :triptime \; 382. \end{array} \right\}.$$

$$\mathcal{S}_{fare} = \left\{ \begin{array}{l} :trans_1 \; :fare \; 7. \\ :trans_1 \; :pickupTime \; "2013-01-0115:11:48". \end{array} \right\}.$$

The temporal context of each event recorded in each stream is specified by a triple with a temporal predicate, i.e., :*pickupTime* or :*dropoffTime*. Note that, the temporal context can be encoded differently in a certain implementation. For instance, the implementations of C-SPARQL [5] and CQELS [6] use the system timestamp at the time when an event (i.e. a triple) arrives to the system. This timestamp is encoded as the temporal context to determine the order of stream elements to be processed in the engine. In the scope of this paper, we assume the stream elements are totally ordered. This assumption might lead to a limitation that there cannot be two taxi rides with the same pickup time to guarantee a fare to be unambiguously connected with a ride. In practice, this limitation can be overcome by using suitable time-unit of the timestamps. Most of such systems define the *sliding-window* operators which are used to extract a finite set of RDF triples as an RDF graph from an RDF stream based on a certain window. The idea of extracting an RDF graph from an unbounded RDF stream is to be able to apply SPARQL query operators, e.g., SPARQL 1.1, so that a continuous query language on RDF stream can inherit the SPARQL grammars. Along this line, the definitions of such window operators on RDF streams are adopted from window operators on relational streams of CQL [14]. Consequently, the semantics of a continuous query on RDF streams are defined as a composition of such query operators. For example, with the above data, a query based on currently agreed syntaxes of RSP community is illustrated as below. This query is used to continuously report *"hourly riding rate of **active taxies** of last 1000 payment transactions"* whereby **active taxis** are the taxis that reported picking-ups within 2 h and dropping-offs within last 1 h.

```
SELECT ?taxi (AVERAGE(?fare/(?tripTime/3600)) AS ?hourlyRate)
FROM  NAMED WINDOW :W1 ON  nyctaxi:fare [COUNT 1000]
FROM  NAMED WINDOW :W2 ON  nyctaxi:pickup
                           [RANGE PT2H @:pickupTime]
FROM  NAMED WINDOW :W3 ON  nyctaxi:dropoff
                           [RANGE PT1H @:dropoffTime]
WHERE {
  WINDOW :W1{ ?trans :fare ?fare. ?trans :pickupTime ?pTime}
  WINDOW :W2{ ?ride :taxi ?taxi. ?ride :pickupTime ?pTime}
  WINDOW :W3{ ?ride :triptime ?tripTime}
  }
GROUP BY  ?taxi
```

Example query on NYC taxi data SPARQL-like continuous query

To evaluate this query, a physical continuous query pipeline is translated as illustrated in Fig. 1. At the root of this query pipeline, the aggregation operator (AVERAGE) consumes inputs from the

[3] https://www.w3.org/community/rsp/.

[4] http://chriswhong.com/open-data/foil_nyc_taxi/.