Challenges in Linked Stream Data Processing: A Position Paper

Danh Le-Phuoc, Josiane Xavier Parreira, and Manfred Hauswirth

Digital Enterprise Research Institute,
National University of Ireland, Galway
Galway, Ireland
{danh.lephuoc,josiane.parreira,manfred.hauswirth}@deri.org

Abstract. Recently, there has been efforts in lifting the content produced by stream sources, e.g. sensors, to a semantic level. In particular, there is ongoing work in representing stream data following the standards of Linked Data, creating what it is called Linked Stream Data. The advantages of Linked Stream Data are manyfold: adding semantics allows the search and exploration of sensor data without any prior knowledge of the data source, and using the principles of Linked Data facilitates the integration of stream data to the increasing number of data collections that form the Linked Open Data cloud, enabling a new range of applications.
However, the highly dynamic and temporal nature of Linked Stream Data poses many challenges in making Linked Stream Data a reality that users and applications can benefit from. In this position paper we address the challenges in Linked Stream Data processing. We will focus on data representation and storage, query model and query processing, highlighting the main differences compared to Linked Data processing and looking at the approaches that currently address these challenges, showing what has been done and what is still needed, suggesting ideas for future research.

Keywords: Linked stream, data storage, query processing, position paper

1 Introduction

Stream data sources, in particular sensors, are very popular nowadays and can be found everywhere, for instance in mobile phones (accelerometer, compass, etc.), in weather observation stations (temperature, humidity, etc.), in the health care domain (heart rate, blood pressure monitors, etc.), in devices for tracking people's and object's locations (GPS, RFID, etc.), and in the Web at large, with online communities services such as Twitter and Facebook delivering real time data on various topics (RSS or Atom feeds, etc.), where users play the role of *citizen sensors* [13].

Sensed data is often archived or streamed as raw data, but rarely associated with enough metadata describing its meaning. Meaning of sensor data includes

the feature of interest, the specification of measuring devices, accuracy, measuring condition, scenario of measurements, location, etc. Such metadata is essential for search and exploration when the user is confronted with large numbers of sensors and gigabytes of sensor data. The lack of metadata also makes the integration of sensor data with other data sources a difficult and labour-intensive task.

There have been a lot of efforts in employing Semantic Web technology to semantically enrich sensor data [8, 14, 16, 18, 21]. In order to allow easy integration with other data sources available in Linked Open Data (LOD) cloud, they suggest that sensor data sources should be published following the Linked Data principles [6] which, among other things, makes the data accessible through a user-friendly URI, creating what is called Linked Data Stream [17]. However, the state-of-the-art Semantic Web technologies are inadequate for enabling Linked Data Stream processing, due to the highly dynamic and temporal aspects of the data.

In this paper we address the challenges of Linked Stream Data processing, focusing on data representation and storage, query model and query processing. We highlight the main differences compared to Linked Data processing which prevent standard techniques to be directly applied. Then, we move on the approaches that currently address these challenges, showing what has been done and what is still needed, suggesting ideas for future research. The remainder of this paper is organized as following. The section 2 focuses on the data representation and the need of new query models. The query processing and integration with other data sources is addressed in section 3. Section 4 concludes the paper and gives some final remarks on the topic.

2 Data Representation and Query Model

Linked Stream Data follows the standards of Linked Data, therefore we believe that it should be represented based on RDF, a widely used standard for Linked Data. A RDF representation of stream data, or RDF Stream, extends RDF by adding temporal information. There is already ongoing work that follows this principle: for stream data, CQL [2] defines a relational model as a bag of (possibly infinite) timestamped tuples. For RDF data, the counterpart of tuples are triples, so approaches like StreamingSPARQL [7] and C-SPARQL [5] suggest to add temporal labels to RDF triples to represent stream data as RDF Stream. In a similar way, efforts like [19] suggest to annotate RDF triples with temporal information. However, since there is no RDF standard that supports temporal data, different approaches diverge in their representation. To overcome this, we suggest a general representation that applies RDF temporal notations [11] for representing RDF Stream. With these notations, a RDF Stream data can be denoted as a RDF temporal graph which is a set of temporal triples, the counterpart of timestamped tuples. Adapting current approaches to this general representation should be straightforward.

With the RDF Stream defined, we now need to model queries over Linked Stream Data. Similar to stream data, queries are expected to be continuous, i.e. they are likely to be valid for a certain time period. We suggest a query model based on CQL. CQL consists of query fragments inherited from relational

query models, plus three new data mappings operators: relational-to-relational mapping, stream-to-relational mapping, and relational-to-stream mapping. Following the same idea, we suggest to define operators to map RDF temporal graphs to RDF graphs and vice versa. The idea of "snapshots" of RDF temporal graphs enables the creation of finite RDF graphs from a temporal graph [11]. For this, sliding windows operators are defined over RDF streams as follows: as RDF Streams can be mapped to RDF fragments, a query model for RDF Stream can be built by extending SPARQL's query pattern. By employing sliding window operators, a window-based graph pattern can be added to SPARQL [15] to enable continuous query on RDF Stream. URIs are assigned to RDF Stream data, as suggested by Sequeda and Cochor [17]. Assigning URIs to RDF streams not only allows to access the RDF streams as materialized data but also enables the query processor to treat the RDF streams as RDF nodes, such that other SPARQL query patterns can be directly applied.

The suggested query model is simple yet quite powerful. To demonstrate it we use the following query example: "whenever a car is within 2km of a junction for which a speed sensor and a traffic camera is available, report the car's average speed and camera image". For that we assume we have sensors streaming images captured by traffic cameras and sensors that can track the speed of cars passing by. We also assume that we have these two types of sensors allocated along different streets, and that cars contain GPS sensors that can stream the car's current location. Finally, there is a metadata graph `http://sensors.deri.org/metadata` containing all other information about these sensors, such as geographic location. Figure 1 shows the example query written using SPARQL query patterns and window-based graph patterns, where `http://sensors.deri.org/streams/mygps/` is the URI of the GPS location stream and *spatial:distant* is a built-in function returning the distant between two coordinates in kilometers.

```
SELECT  ?junctionName ?snapShot AVERAGE(?speed) as avgSpeed
FROM NAMED <http://sensors.deri.org/streams/mygps/> [now] as ?gps
FROM NAMED ?trafficcamera [now] as ?junctionImage
FROM NAMED ?trafficsensor [RANGE 30 seconds] as ?carSpeed
FROM NAMED <http://sensors.deri.org/metadata>
WHERE {
   GRAPH ?junctionImage {?camera cam:hasSnapShot ?snapShot}
   GRAPH ?gps {?car geo:lat ?carLat.?car geo:long ?carLon}
   GRAPH ?carSpeed {?car traffic:passbySpeed ?speed}
   GRAPH <http://sensors.deri.org/metadata>  {
    ?trafficcamera geo:locatedAt ?junctionLoc.
    ?traffisensor  geo:locatedAt ?junctionLoc.
    ?junctionLoc   geo:lat ?juncLat.
    ?junctionLoc   geo:long ?juncLong.
    ?junctionLoc   geo:name ?junctionName.
    FILTER {spatial:distant(?carLat,?carLon,?juncLat,?juncLong)<=2}
   }
}
GROUP BY ?speed
```

Fig. 1. Example of a continuous query over Linked Stream Data.

The query first gets the car's current location (given by the GPS), and joins it with the graph containing the metadata, which provides the identifiers for the speed and traffic camera sensors at the junctions. Since the URIs of the traffic camera streams and the traffic sensor streams needed are unknown and subject

to change (since they depend on the car's location), they are represented as variables in the graph query pattern. The sliding-window operators, *[NOW]* for current snapshot and *[RANGE]* for snapshots within a time range, are applied over the continuous traffic camera and speed stream. The result of these operators are materialized and represented as RDF graphs that can be processed by the SPARQL query processor. Details of the proposed formalization for RDF Streams and the model for continuous query over Linked Stream Data are presented in [12].

3 Query Processing

Even though our proposed query model allows queries to be executed using standard query processors for triplestores, the execution is very inefficient, since they do not support continuous queries. That means that each query would have to be repeatedly issued as often as the updates on the streams, for as long as the query is valid, every time checking if the new values satisfy the query's conditions. In some cases, as in the example query from previous section, the query is valid for a long period of time, which make this approach prohibitive. StreamingSPARQL has addressed this issue by having translation rules that translate continuous queries to SPARQL algebras and sliding-window operators. Although it gives a solution for handling continuous queries, this approach is still quite inefficient, since triplestores are mostly based on relational database storage, which are proved to be inefficient for data with high update rates [3]. To solution proposed by C-SPARQL combines triple stores with data stream management systems (DSMS). When a continuous query arrives, it is first split into static and dynamic parts. The framework orchestrator loads bindings of the static parts into relations, and the query is executed by processing the stream data against these relations. Even though it is more efficient than the method used in StreamingSPARQL, C-SPARQL does not take advantage that Linked Stream Data can be combined with existing Linked Data collections. Both stream data management and triple storage systems are used independently as "black boxes", therefore C-SPARQL may miss out on additional potential for optimization over the unified data. Both StreamingSPARQL and C-SPARQL solutions are not very novel, but they rather extend/combine existing query processing approaches. We suggest to look deeper into important aspects of continuous query processing over integrated stream and non-stream data, such as memory consumption, caching, and query optimization, to derive more efficient solutions.

A major issue in continuous query processing is memory consumption. It is common that Linked Stream Data processing involves a large amount of data that is likely not to fit into main memory. Therefore, intermediate results need to be stored on disk and later reloaded for further processing. Since disk reads/writes are generally expensive minimizing such operations becomes very important. One approach to the problem is to apply *dictionary encoding*, which is commonly used by triplestores [1, 9, 10]. Dictionary encoding maps node values (which can be URIs, blank nodes or literal string values) to integer values, which reduces the size of each triple, allowing more triples to fit into memory. The drawback of dictionary encoding is that the cost of keeping the dictionary of mappings might be too high for very dynamic data.

In applications that involves a combination of many data sources, especially if some of them are not stream sources, the performance of the query processor can be greatly improved if some of the intermediate results are cached, for instance, for the input data that do not change very often during the duration of the query. Results reported in [12] demonstrate the benefits of caching. Even for the fast changing stream sources, we can think of caching policies for intermediate results that are shared among multiple queries. In both cases, a mechanism to decide when and what to cache that adapts to the changes of the data is needed [4].

Traditional relational databases are equipped with query optimizers which are responsible for finding the best execution plan. Such feature is also desirable in Linked Stream Data processing. A query optimizer typically computes the optimal query plan in the compiling phase using the statistical distribution of the data. However, in context of stream data, this optimizing technique does not yield satisfying results, as the distribution of the data changes during run-time. The CQELS system [12] provides an adaptive cost-based query optimization algorithm for dynamic data sources, such as stream data. This query optimizer retains a subset of the possible execution plans and, at query time, updates their respective costs and chooses the least expensive one for executing the query at this given point in time. However, depending on the query, the search space for finding the optimal plan might be too big, so heuristics are needed. One suggestion would be to break the query into simpler sub-queries and optimize them separately. In addition, combining caching and query optimization could lead to improvements in the performance.

Further ideas to improve continuous query processing are controlling the sampling rate of the input stream data and building stochastic/statistical model for predicting data series. Both ideas aim at reducing the number of data that needs to be retrieved for query evaluation. The former suggest to sample the data from stream source in a slower rate than the data is produced. While this results in lost of information, there are many applications in which sampling might suffice. The latter consists in modeling the stream source, such that the data values can be predicted, avoiding access to the stream source. In both cases, a combination of prior knowledge derived from historical data, human knowledge in the form of processing rules and reasoners is needed. In particular, reusing domain knowledge represented as ontologies and rules, and performing reasoning in continuous query processing is an open and interesting research area [20].

4 Conclusion

This position paper provides an overall picture of a new emerging research area, Linked Stream Data processing. We have addressed the main challenges in this area regarding data representation and storage, query model and query processing. We have shown why standard Semantic Web technologies can not be directly applied, and highlighted research that is currently being carried out to solve these issues. However, there is still several open issues and our paper have also suggested ideas for future research.

5 Acknowledgements

This work has been supported by the Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-2) and the Irish Research Council for Science, Engineering and Technology (IRCSET).

References

1. D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach. Scalable semantic web data management using vertical partitioning. In *VLDB*, pages 411–422, 2007.
2. A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2):121–142, 2006.
3. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS*, pages 1–16. ACM, 2002.
4. S. Babu, K. Munagala, J. Widom, and R. Motwani. Adaptive caching for continuous queries. In *ICDE*, pages 118–129, 2005.
5. D. F. Barbieri, D. Braga, S. Ceri, and M. Grossniklaus. An execution environment for c-sparql queries. In *EDBT*, pages 441–452. ACM, 2010.
6. C. Bizer, T. Heath, and T. Berners-Lee. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009.
7. A. Bolles, M. Grawunder, and J. Jacobi. Streaming sparql extending sparql to process data streams. In *ESWC*, pages 448–462, 2008.
8. E. Bouillet, M. Feblowitz, Z. Liu, A. Ranganathan, A. Riabov, and F. Ye. A semantics-based middleware for utilizing heterogeneous sensor networks. In *3rd IEEE international conference on Distributed computing in sensor systems*, pages 174–188, 2007.
9. E. I. Chong, S. Das, G. Eadon, and J. Srinivasan. An efficient SQL-based RDF querying scheme. In *VLDB*, pages 1216–1227, 2005.
10. J. B. et al. Sesame: An architecture for storing and querying rdf data and schema information. In *Spinning the Semantic Web*, 2003.
11. C. Gutierrez, C. A. Hurtado, and A. Vaisman. Introducing Time into RDF. *IEEE Transactions on Knowledge and Data Engineering*, 19:207–218, 2007.
12. D. Le-Phuoc, J. X. Parreira, M. Hausenblas, and M. Hauswirth. Continuous query optimization and evaluation over unified linked stream data and linked open data. Technical Report DERI-TR-2010-09-27, DERI, IDA Business Park, Lower Dangan, Galway, Ireland, 9 2010.
13. M. Nagarajan, K. Gomadam, A. P. Sheth, A. Ranabahu, R. Mutharaju, and A. Jadhav. Spatio-temporal-thematic analysis of citizen sensor data: Challenges and experiences. In *WISE*, pages 539–553, 2009.
14. H. Patni, C. Henson, and A. Sheth. Linked sensor data. In *CTS*, 2010.
15. J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3):1–45, 2009.
16. A. Rodríguez, R. McGrath, Y. Liu, and J. Myers. Semantic management of streaming data. In *SSN*, pages 80–95, 2009.
17. J. F. Sequeda and O. Corcho. Linked stream data: A position paper. In *SSN*, pages 148–157, 2009.
18. A. P. Sheth, C. A. Henson, and S. S. Sahoo. Semantic Sensor Web. *IEEE Internet Computing*, 12(4):78–83, 2008.
19. U. Straccia, N. Lopes, G. Lukacsy, and A. Polleres. A general framework for representing and reasoning with annotated semantic web data. In *AAAI*, 2010.
20. G. Unel and D. Roman. Stream reasoning: A survey and further research directions. In *FQAS*, pages 653–662, 2009.
21. K. Whitehouse, F. Zhao, and J. Liu. Semantic Streams: A Framework for Composable Semantic Interpretation of Sensor Data. In *EWSN*, pages 5–20, 2006.