Towards a distributed, scalable and real-time RDF Stream Processing engine

Xiangnan Ren[1,2,3]

[1] ATOS - 80 Quai Voltaire, 95870 Bezons, France
{xiang-nan.ren}@atos.net
[2] ISEP - LISITE, Paris 75006, France
[3] LIGM (UMR 8049), CNRS, UPEM, F-77454, Marne-la-Vallée, France

Abstract. Due to the growing need to timely process and derive valuable information and knowledge from data produced in the Semantic Web, RDF stream processing (RSP) has emerged as an important research domain. Of course, modern RSP have to address the volume and velocity characteristics encountered in the Big Data era. This comes at the price of designing high throughput, low latency, fault tolerant, highly available and scalable engines. The cost of implementing such systems from scratch is very high and usually one prefers to program components on top of a framework that possesses these properties, e.g., Apache Hadoop or Apache Spark. The research conducting in this PhD adopts this approach and aims to create a production-ready RSP engine which will be based on domain standards, e.g., Apache Kafka and Spark Streaming. In a nutshell, the engine aims to i) address basic event modeling - to guarantee the completeness of input data in window operators, ii) process real-time RDF stream in a distributed manner - efficient RDF stream handling is required; iii) support and extend common continuous SPARQL syntax - easy-to-use, adapt to the industrial needs and iv) support reasoning services at both the data preparation and query processing levels.
Keywords: Stream Processing, Distributed Computing, Semantic Web, RDF, RSP

1 Problem statement

Nowadays, the RDF data format is getting more and more popular in the Internet of Things (IoT) ecosystem, i.e., data produced by sensors or other devices are either directly represented as RDF triples or trasnformed into this standard. The heterogeneous nature of IoT data sources potentially presents multiple challenges for the implementation of real-time RDF Stream Processing (RSP) services. These challenges refer to four distinct problematics [16,25,9,2]:

1. functionality support: from a user friendliness perspective, the RSP engine is supposed to support common SPARQL syntax.
2. correctness of output: the query answers produced by the system are expected to be correct.

3. performance: like any other stream processing system, high throughput, low response latency, scalability are considered as the performance criteria for RSP engine.
4. reasoning: correctness and completeness of query answers may depend on computed inferences.

To address the above-mentioned problems, we can identify two categories of RSP engines which have designed in the last few years. The category of **centralized** systems, including C-SPARQL [5], CQELS [15], SPARQL$_{stream}$ [7], ETALIS [3]/EP-SPARQL[1] have lead the way in the direction of RDF stream processing but are limited in terms of the amount of events that can handle. This has motivated a second generation of RSP which are **distributed**, e.g., CQELS-Cloud [17], Katts [11] and Distributed-Etalis. Although reaching better throughput and latency performances, these systems do not integrate the state of the art approaches that are currently guaranteeing fault tolerance, highly availability and scalability and which can enforce the system's robustness and correctness.

2 Problem Relevancy and Motivation

Stream processing in general is one of the hottest topics in Big data. It is currently supporting analytics functionalities that have not been considered before. This is mainly due to the popularization of Web technologies (i.e., the pipeline that provides streams) as well as advancements in computing (i.e., emergence of distributed frameworks that facilitate the design of parallel computations).

RDF data is relevant in this streaming context because i) it is a popular graph format equipped with an expressive SPARQL query language, ii) it is anchored in the Web with its wide use of URIs and iii) it has the ability to support reasoning services.

The project within which my PhD research is taking place requires an important subset of the Semantic Web standards, namely RDF, RDFS and OWL considering facilities to represent and reason graph data and knowledge, SPARQL as a well-established query language and Linked Open Data to access external knowledge. The project concerns industrial water resources management. The events emitted from various sensors correspond to pressure, flow, chlorine, temperature, turbidity, etc. values and need to be processed almost in real-time to trigger system alerts. The analysis of these events is especially valuable if joined with external, contextual data such as the geographical properties where the sensor is situated.

Of course, our goal is to design a generic RSP engine that can adapt easily to use cases concerned with other domains. Intuitively, the goal is to seamlessly integrate novel ontologies, data/knowledge repositories and sets of queries within a highly distributed, reasoning-enabled, continuous query and complex event processing system. This aspect is particularly important since the PhD thesis is funded by a large IT company which envisions to have commercial activities using this system.

[1] EP-SPARQL, a wrapper of ETALIS to support SPARQL-like syntax

3 Related work

In the last few years, a variety of RDF stream processing engines have been proposed. As mentioned in section 1, the design of RSP needs to cover three aspects [16,2]: functionality support, output correctness and performance. Engine performance is the most concerned issue, since the fine-grained and schema-free nature of RDF data could lead to intensive join operations on query task. For the convenience of illustration, we divide the RSP engines into two categories: centralized and distributed engines.

Centralized RSP engines: currently, C-SPARQL, CQELS, SPARQL$_{stream}$, and ETALIS/EP-SPARQL are popular centralized RSP engines. All of them are developed to run on a single machine.

Distributed RSP engines: CQELS-Cloud is the first distributed RSP system which mainly focuses on the engine elasticity and scalability. The whole system is based on Storm[2]. Firstly, CQELS-Cloud compresses the incoming RDF streams by dictionary encoding in order to reduce the data size and the communication in the computing cluster. Then, to overcome the performance bottlenecks on join tasks, the authors propose a *parallel multiway join* based on probing sequence, which is inspired by the MJoin algorithm [21]. However, CQELS-Cloud uses the conventional RDF triple $(subject, predicate, object)$ as its data model and has not provided any event model yet. Furthermore, to the best of our knowledge, CQELS-Cloud is not completely open source, and current CQELS-Cloud does not allow external users to define customized queries.

Katts is another RSP engine based on Storm, which applies graph partitioning [3] to optimize the message exchanging for cluster computing. Based on our evaluation, we can say that Katts allows a limited amount of query operators and queries with only-streaming data sources. Besides, neither query algebra optimization nor SPARQL syntax support have been considered. Finally, the implementation of Katts remains at the stage of scientific prototype.

Distributed-ETALIS is a distributed Complex Event Processing (CEP) engine based on Storm and Kafka [4]. Both Distributed-ETALIS and its centralized version ETALIS implement a rule-based DSL for event detection (pattern matching). It seems that ETALIS team has stopped the software maintenance. We also meet a serious scalability problem in our preliminary evaluation of ETALIS.

A distributed RSP system must rely on a generic distributed stream processing framework such as Storm, Spark Streaming [23,4,24] and Flink [5]. These frameworks have proven their successes in countless scientific and industrial applications. Comparing to Spark (Streaming) and Flink, Storm provides a relatively low-level programming API to allow generic real-time service design. Spark Streaming, one of the principal components of Spark ecosystem, is a near real-time distributed computing framework. Current Spark-Streaming is based on

[2] `http://storm.apache.org`

[3] `http://glaros.dtc.umn.edu/gkhome/views/metis`

[4] `http://kafka.apache.org`

[5] `https://flink.apache.org`

micro-batch execution mechanism, and provides the sub-second delay. Flink is another popular massively parallel data processing engine which supports real-time data processing and CEP. Due to the enrichment and the maturity of the platform ecosystems, we choose Spark Streaming as the framework of our RSP engine.

4 Research questions

Given the context of this PhD thesis, the general research question is: *How to efficiently query and reason over massive real-time RDF events data in a distributed computing environment?* To answer this question, the following four aspects have to be considered:

- **Q1 - Distributed, robust RDF stream query processing:** we consider the design of an architecture for the processing of RDF streams that is highly available, tolerates failures, scales and guarantees high throughput and low latencies. No RSP engine possesses all these properties and present performance figures comparable to non-RDF state of the art engines. In particular, these last engines are generally designed using a distributed pub/sub messaging system (e.g., Apache Kafka) and a dedicated stream engine (e.g., Apache Flink, Storm, Spark streaming, Beam). Replying to this question implies to adapt RDF peculiarities to this rapidly evolving ecosystem.
- **Q2 - Compression and reasoning:** processing (i.e., querying and reasoning) in a real-time manner over complex graph-based events implies to optimize all computation aspects. We consider that a decompression-free querying approach tightly connected to a semantic-based ontology encoding can guarantee these properties.
- **Q3 - Extending continuous SPARQL query toward complex event processing capacities:** we have identified some practical use cases that are requiring continuous query features that we have not encountered in available continuous SPARQL query languages, e.g., session-based windows. These laguage extensions have to be integrated in our declarative query language.
- **Q4 - How to evaluate RSP engines:** we need a way to evaluate the system that is currently being designed. This implies to identify, experiment and evaluate streaming benchmarks and engines. A set of performance metrics also need to be defined to evaluate distributed RSP engines in a cluster environment.

5 Preliminary results

Some preliminary results have obtained for Q4 of section 4. The main idea was to evaluate three centralized RSP systems, i.e. C-SPARQL, CQELS, ETALIS. This evaluation defines a complete set of performance metrics to measure query latency and memory consumption by varying stream rate, window size, number of streams , etc. The evaluation of distributed RSP engines are in progress, some

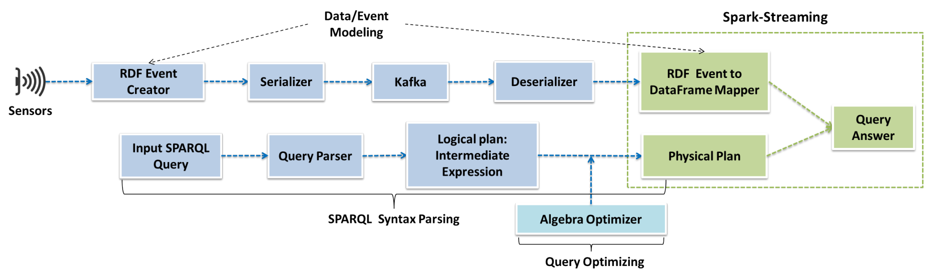

Fig. 1: Engine architecture

performance metrics need to be added, such as measure the query latency and engine throughput by varying the number of machines. We have currently tested Katts on Amazon EC2 [6] by using the real-world data. The evaluation results of Katts are kept for future comparisons with our system. This part of work gives us a deep understanding on existing RSP engines and continuous SPARQL query language features. This has helped me on the design and the implementation of our own RSP engine.

Recently, I started to develop a prototype [7] that will run continuous SPARQL queries on Spark Streaming and Kafka. In Figure 1, I provide a high-level view of the system's architecture. The data obtained from sensors or other sources are first encoded as RDF events. Next, the obtained RDF event streams continue to be serialized into binary format and transmitted to Kafka. Then, Spark-Streaming concurrently receives, caches and deserializes the incoming binary stream. Finally, the system applies the precompiled optimized logical plan to proceed the query execution.

6 Approach

In this section, I present how I intend to address the Q1, Q2 and Q3 research questions presented in section 4 as well as how I will validate the system efficiency.

Q1 addresses to three subquestions: *i) Data/Event Modeling*. Firstly, the system aims to support two RDF formats for data updating: triple (i.e. (s, p, o)) by triple and event by event. Basically, an RDF triple can be regarded as the simplest RDF graph pattern. I use Sesame [6] to convert the sensors data into *RDF events*. An RDF event is essentially a set of triple patterns. To identify each event and its belonging stream source, the representation of triple pattern (s, p, o) is extended. I.e., an event is formed as $e = (strId, evtId, t, \{(s_n, p_n, o_n)\}_{n=1,\ldots,N})$ [8]. *ii) SPARQL syntax supporting*. To support distributed querying on Spark

[6] https://github.com/renxiangnan/RSP_Evaluation_Results
[7] https://github.com/renxiangnan/rsp
[8] A Java/Scala collection object

Streaming, the incoming RDF event streams need to be cached in window operator and converted into Spark pre-defined data structure. [14] gives a road map to choose an appropriate Spark distributed data collection. I create a native parser to parse SPARQL query into the Spark relational operators. The parser takes Sesame to transform the SPARQL query operators into an intermediate infix expression, namely, *logical plan.* The logical plan will be continuously parsed into a physical plan (i.e. algebra tree) to proceed the query execution.

Due to the *real-time* aspect in streaming context, some optimization techniques based on data preprocessing become hardly applicable, e.g. data indexing, vertical partitioning [1], property tables [22] etc. My current research mainly focuses on query algebra rewriting. Based on existing work [18,13,19,8,20], I plan to redesign a three-layers optimization strategy to simplify the query algebra, reduce the overhead of the aggregate operators and adjust the join order of triple patterns.

Q2 concerns compression and reasoning aspects. I approach goes along the work of ERI[12] and RDSZ[10] but aims to go one step further in terms of compression. That is, we aim to adopt a two structures approach, one containing graph event patterns and the other one storing data bindings associated to these graph patterns. Graph patterns will be represented as compact graph signature that will facilitate the efficient discovery of similar signatures. Moreover, the elements of these signatures will correspond to semantic-aware numerical encoding of underlying ontologies. This approach will support reasoning at both materialization (to enrich graph events that can potentially satisfy a set of continuous queries) and query reformulation (for optimization and satisfiability purposes) processing.

The continuous SPARQL query language we have implemented so far is inspired by C-SPARQL. It supports the definition of different window forms, e.g., fixed or sliding windows, via keywords which are permitted in queries. We have identified an additional session-based window form that is required in some practical use cases. Hence, the approach to address **Q3** will consist in defining the proper semantics for additional continuous SPARQL query clauses, implementing this semantics within our distributed streaming engine and integrating optimization for this extended query languages, e.g., to support efficient query reformulation. Finally, this novel approach will be heavily tested and evaluated against practical use cases and data sets.

7 Evaluation plan

Our system will be evaluated on both synthetic and real-world data sets. Concerning the synthetic data sets, we have already identified, experimented and even extended well-established benchmarks during our work of Q4. The real-world experimentation will be directly related to the water resources management use case. It consists of values obtained from real sensors. We aim to test the system live as well as replay old values and thus test whether our system is able to discovery peculiar situations that we known have occurred in the past.

The evaluation will be conducted both in local mode and distributed mode, with respect to the following outline.

Functionality support. Since our sytem aims to fully automatize the query execution, we need to implement all the common query operators ((*BGP* join, filter, aggregations, etc.). **Correctness.** On both synthetic and real-world data sets, we will be able to qualify the correctness of the provided answers. On our water resources management, we already know that correctness implies some reasoning. **Performance**. Two performance aspects need to be studied: Latency and throughput. Latency refers to the wall-clock time consumed by the engine on each query execution. Throughput depicts that *how many RDF triples can be processed in a unit time.* These two performance metrics gives us insights into whether the system could handle the target scenario or not. The engine throughput and query latency are recorded by varying the input stream rate (events/second) and the number of cluster nodes.

8 Reflections

This PhD thesis is now at an early stage and preliminary results are promising. I still have a lot to design and implement but I'm getting closer to the point where room for thorough experimentation, evaluation, exploration and innovation will be possible. In the near future, I'm planning to propose some trade-off between materialization and query rewriting as well as novel optimization for RDF stream processing, e.g. querying over compressed data.

References

1. D. J. Abadi, A. Marcus, S. Madden, and K. J. Hollenbach. Scalable semantic web data management using vertical partitioning. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, pages 411–422, 2007.
2. M. I. Ali, F. Gao, and A. Mileo. Citybench: A configurable benchmark to evaluate RSP engines using smart city datasets. In *The Semantic Web - ISWC 2015.*
3. D. Anicic, P. Fodor, S. Rudolph, R. Stühmer, N. Stojanovic, and R. Studer. A rule-based language for complex event processing and reasoning. In *Web Reasoning and Rule Systems*, pages 42–57, 2010.
4. M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, and M. Zaharia. Spark SQL: relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD*, pages 1383–1394, 2015.
5. D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, and M. Grossniklaus. C-SPARQL: SPARQL for continuous querying. In *Proceedings of the 18th International Conference on World Wide Web*, pages 1061–1062, 2009.
6. J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: An architecture for storin gand querying RDF data and schema information. In *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential [outcome of a Dagstuhl seminar]*, pages 197–222, 2003.
7. J. Calbimonte, Ó. Corcho, and A. J. G. Gray. Enabling ontology-based access to streaming data sources. In *The Semantic Web - ISWC 2010*, pages 96–111, 2010.

8. D. Chatziantoniou, M. O. Akinde, T. Johnson, and S. Kim. The md-join: An operator for complex OLAP. In *Proceedings of the 17th International Conference on Data Engineering*, pages 524–533, 2001.
9. D. Dell'Aglio, J. Calbimonte, M. Balduini, Ó. Corcho, and E. D. Valle. On correctness in RDF stream processor benchmarking. In *The Semantic Web - ISWC 2013*, pages 326–342, 2013.
10. J. D. Fernández, A. Llaves, and Ó. Corcho. Efficient RDF interchange (ERI) format for RDF data streams. In *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part II*, pages 244–259, 2014.
11. L. Fischer, T. Scharrenbach, and A. Bernstein. Scalable linked data stream processing via network-aware workload scheduling. In *Proceedings of the 9th International Workshop on Scalable Semantic Web Knowledge Base Systems*, pages 81–96, 2013.
12. N. F. García, J. Arias-Fisteus, L. Sánchez, D. Fuentes-Lorenzo, and Ó. Corcho. RDSZ: an approach for lossless RDF stream compression. In *The Semantic Web: Trends and Challenges - 11th International Conference, ESWC 2014*, pages 52–67, 2014.
13. H. Kim, P. Ravindra, and K. Anyanwu. From SPARQL to mapreduce: The journey using a nested triplegroup algebra. *PVLDB*, pages 1426–1429, 2011.
14. H. Naacke, O. Curé, and B. Amann. SPARQL query processing with Apache Spark. *ArXiv e-prints*, 2016.
15. D. L. Phuoc, M. Dao-Tran, J. X. Parreira, and M. Hauswirth. A native and adaptive approach for unified processing of linked streams and linked data. In *The Semantic Web - ISWC 2011*, pages 370–388, 2011.
16. D. L. Phuoc, M. Dao-Tran, M. Pham, P. A. Boncz, T. Eiter, and M. Fink. Linked stream data processing engines: Facts and figures. In *The Semantic Web - ISWC 2012*, 2012.
17. D. L. Phuoc, H. N. M. Quoc, C. L. Van, and M. Hauswirth. Elastic and scalable processing of linked stream data in the cloud. In *The Semantic Web - ISWC 2013*, pages 280–297, 2013.
18. P. Ravindra. Towards optimization of RDF analytical queries on mapreduce. In *Workshops Proceedings of the 30th International Conference on Data Engineering Workshops, ICDE 2014*, pages 335–339, 2014.
19. P. Ravindra, H. Kim, and K. Anyanwu. Optimization of complex SPARQL analytical queries. In *Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016*, pages 257–268, 2016.
20. P. Tsialiamanis, L. Sidirourgos, I. Fundulaki, V. Christophides, and P. Boncz. Heuristics-based query optimisation for sparql. In *Proceedings of the 15th International Conference on Extending Database Technology*, EDBT '12, pages 324–335, 2012.
21. S. Viglas, J. F. Naughton, and J. Burger. Maximizing the output rate of multi-way join queries over streaming information sources. In *VLDB*, pages 285–296, 2003.
22. K. Wilkinson. Jena property table implementation. In *SSWS*, pages 35–46, 2006.
23. M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *2nd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'10*, 2010.
24. M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica. Discretized streams: fault-tolerant streaming computation at scale. In *ACM SIGOPS 24th Symposium on Operating Systems Principles, SOSP*, pages 423–438, 2013.
25. Y. Zhang, M. Pham, Ó. Corcho, and J. Calbimonte. Srbench: A streaming RDF/SPARQL benchmark. In *The Semantic Web - ISWC 2012*, pages 641–657, 2012.