Apache Spark and Apache Kafka at the rescue of distributed RDF Stream Processing engines

Xiangnan Ren[1,2,3], Olivier Curé[3], Houda Khrouf[1], Zakia Kazi-Aoul[2], Yousra Chabchoub[2]

[1] ATOS - 80 Quai Voltaire, 95870 Bezons, France
{xiang-nan.ren, houda.khrouf}@atos.net
[2] ISEP - LISITE, Paris 75006, France
{zakia.kazi, yousra.chabchoub}@isep.fr
[3] LIGM (UMR 8049), CNRS, UPEM, F-77454, Marne-la-Vallée, France
olivier.cure@u-pem.fr

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. In this paper, we describe the design of an RSP engine that is built upon state of the art Big Data frameworks, namely Apache Kafka and Apache Spark. Together, they support the implementation of a production-ready RSP engine that guarantees scalability, fault-tolerance, high availability, low latency and high throughput. Moreover, we highlight that the Spark framework considerably eases the implementation of complex applications requiring libraries as diverse as machine learning, graph processing, query processing and stream processing.

1 Introduction

The Resource Description Framework (RDF) is a flexible data format that was originally designed for the Web, and is now gaining popularity in the Internet of Things (IoT). A majority of IoT data are dynamically generated from various sources, e.g., sensors, and require inference services to meet their full analytic potential. This trend leads to the notion of RDF Stream Processing (RSP) which gains more and more attention as a research topic.

Due to the baseline defined by some RSP benchmarks such as LSBench [5] and SRBench [7], modern RSP engines need to address the following aspects: support of SPARQL main operators, output correctness and engine performance. To cope with these fundamental requirements, several centralized engines have been proposed in the last decade, such as C-SPARQL [1] and CQELS [4]. Limited by a centralized design, these systems can hardly deal with the volume growth and velocity increase of RDF streams. Therefore, a distributed solution for RSP is needed to deal with practical workloads. The current distributed RSP engines such as CQELS-Cloud [6] and Katts [2] make a significant progress on engine performance and scalability. Nevertheless, all available distributed RSP systems lack important features, e.g., support for common SPARQL operators and are not ready for production. Moreover, they do not integrate the state of the art approaches that are currently guaranteeing fault tolerance, highly availability which can enforce the system's robustness.

To meet the expectations of Big Data projects, a novel RSP engine is required. Motivated by the WAVES project[1], this work provides insights on the integration of Apache Kafka, a distributed messagging broker, and Apache Spark, a distributed computing framework. These two Big Data frameworks will ensure robustness, reliability and scalability properties.

Applications based on stream processing frequently require different libraries and tools, to (1) perform incremental and iterative tasks, e.g., query processing and machine learning, (2) process graph data models and (3) handle all the streaming machinery, e.g., continuous query, windowing operations. Currently, two distributed stream processing frameworks have been emphasized by their adoption in large industrial projects: Apache Spark and Apache Flink. Spark-Streaming is based on *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.

2 Use case

Our motivating use case concerns the industrial application of water resource network management. It aims to provide real-time analytics over RDF data streams. The observations, also denoted as *events*, are dynamically generated from various sensors and are hence anchored in spatio-temporal analytics. The measures we are considering correspond to pressure, flow, chlorine, temperature and turbidity. Their real-time analysis permit to detect water network anomalies, such as water leaks, and can have important impacts at the economical and environmental levels. Nevertheless, our goal is to design a generic RSP engine that can easily adapt to use cases concerned with other domains. Intuitively, the goal is to seamlessly integrate novel ontologies, data streams and sets of queries within a highly distributed, reasoning-enabled, continuous query and complex event processing system.

3 Architecture

Figure 1 gives a high-level overview of our system architecture, in which we introduce the use of the principal components. The incoming RDF events are dynamically filtered and converted into a compressed RDF serialization. Note that an RDF event is essentially a set of triples. To identify each event and its stream source, the representation of triple pattern (s, p, o) is extended, i.e., an event is formed as a set of triples as $e = (streamID, eventID, t, \{(s_n, p_n, o_n)\}_{n=1,...,N})$

[1] More details at http://waves-rsp.org

where t is the event timestamp. Then, the obtained RDF event streams are continuously sent to the Kafka message broker. We use Kafka to manage incoming event streams. Typically, each pre-defined Kafka *topic* is associated to some specific RDF events (e.g., the event of *flow observation* or *chlorine observation*, etc.). Finally, Spark-Streaming concurrently receives, caches and deserializes incoming data streams.

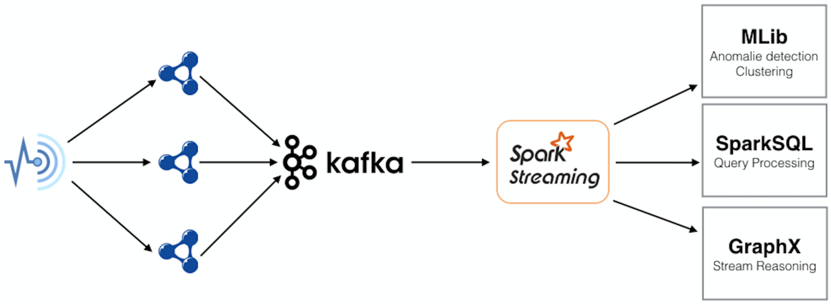

Fig. 1: Water Resource Management Context

We combine Spark SQL, MLib and GraphX with Spark Streaming libraries to form the computing core of our system. It mainly covers the RDF data analytic in two aspects: basic timely SPARQL query processing and data mining over RDF streams. Indeed, the canonical SPARQL query processing with external and contextual data becomes valuable, which may also support stream reasoning. Moreover, the system is also planned to meet the requirement of advanced data analytic, such as classification and anomaly detection. Besides, Spark provides a seamless connection among its available libraries, since they share the same data collection abstraction, i.e. `RDD`. In the following, we present the tasks that highlight the use of these libraries.

Spark Streaming is an extension of the core Spark API and enables near real-time stream processing. We use Spark Streaming as the basic stream processing layer. It receives input data from Kafka, divides and parallelizes the data into batches, which are next processed by Spark Core.

Spark SQL provides a high-level abstraction to support distributed relational operations. The work in [3] gives a road map to choose the appropriate approach for SPARQL query processing on Spark. In our system, we convert the input RDF event into a data collection called `DataFrame` before query execution. Then, we use Sesame API to parse a (continuous) SPARQL query and return the query algebra tree. The obtained algebra tree allows us to reconstruct an equivalent and optimized algebra tree on Spark, namely *logical plan*. Finally, we dynamically generate the code from the *logical plan* for query processing.

MLib is a native Spark library for machine learning. We use MLib to run the data analytics pipeline. One of the target scenario is anomaly detection. Once the event is received by Spark Streaming, we push events to the analytic layer which uses classification algorithms and decision tree. For instance, we compare measures for the same geographical sector on two different dates using a clustering approach such as k-means.

GraphX facilitates parallel graph processing on Spark. We mainly use GraphX to generate semantic-aware dictionaries for our knowledge bases. Intuitively, it enables to seamlessly compute connected components, for instance to compute the transitive closure of some hierarchies or triples involving the `owl:sameAs` property. This computation only requires to provide two RDDs, containing vertices and edges of the RDF graph, to a given function. The output can easily be processed within our Spark core program.

4 Conclusion

RDF stream processing is an emerging area that is still in its infancy and hence requires to conduct much more research. In this work, we present our envisioned RSP system which is based on Spark Streaming and Kafka. These mature ecosystems bring important properties expected in Big Data applications and have proved to ease the design of application requiring libraries as diverse as machine learning, graph computations, query processing and of course stream management. In future work, we plan to extend our query processing with a trade-off between two common reasoning approaches, namely materialization and query reformulation.

5 Acknowledgment

This work is funded by the Fonds Unique Interministériel (FUI #17) through the WAVES project, available at `http://waves-rsp.org`.

References

1. 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 (WWW)*, pages 1061–1062, 2009.
2. 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.
3. H. Naacke, O. Curé, and B. Amann. SPARQL query processing with Apache Spark. *ArXiv e-prints*, 2016.
4. 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 10th International Semantic Web Conference (ISWC)*, pages 370–388, 2011.
5. 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 11th International Semantic Web Conference (ISWC)*, pages 300–312, 2012.
6. 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 12th International Semantic Web Conference (ISWC)*, pages 280–297, 2013.
7. Y. Zhang, M. Pham, Ó. Corcho, and J. Calbimonte. Srbench: A streaming RDF/SPARQL benchmark. In *the 11th International Semantic Web Conference (ISWC)*, pages 641–657, 2012.