Platform-Agnostic Execution Framework Towards RDF Stream Processing

Danh Le-Phuoc[1], Minh Dao-Tran[2], Chan Le Van[1], Anh Le Tuan[1], Manh Nguyen Duc[1], Tuan Tran Nhat[1], and Manfred Hauswirth[3]

[1] Insight Centre for Data Analytics, National University of Ireland, Galway
{danh.lephuoc,chan.levan,anh.letuan,ducmanh.nguyen,tuan.trannhat}@insight-centre.org
[2] Institute of Information Systems, Vienna University of Technology
Favoritenstraße 9-11, A-1040 Vienna, Austria
dao@kr.tuwien.ac.at
[3] Institut für Telekommunikationssysteme, Technische Universität Berlin, Berlin, Germany
manfred.hauswirth@tu-berlin.de

Abstract. This paper presents a platform-agnostic execution framework towards RDF Stream Processing (RSP), called, Continuous Query Evaluation over Linked Streams (CQELS). CQELS framework is the core infrastructure for CQELS engine, one of the initiatives of the RSP community. Moreover, the framework is platform-agnostic to build other RSP engines that work on embedded devices as well as cloud infrastructure. The development roadmap towards platform independent is also discussed in this paper.

Keywords: RDF Stream Processing, Linked Stream Data, Continuous Query, Execution Framework

Overview of CQELS Framework

CQELS Framework provides a platform-independent infrastructure to implement RSP engines for computing continuous queries expressed as an extension of SPARQL 1.1 [7], called CQELS-QL. The abstract architecture [9] illustrated in Figure 1 accepts RDF streams and RDF datasets as inputs and returns RDF streams or relational streams in the SPARQL Result format [6] as output. The output RDF streams can be fed into any RSP engine, and the relational streams can be used by other relational stream processing systems.

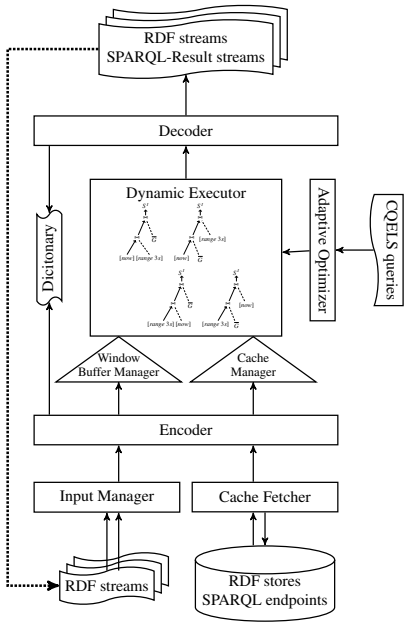

Fig. 1: Native and Adaptive Execution Framework of CQELS.

The execution framework is composed of native and adaptive components organised in the processing flow as follows:

- The stream data is pushed to the Input Manager and is then encoded by the Encoder into a normalised representation. RDF datasets, which can be hosted in a local RDF store or remote RDF stores with SPARQL endpoints, are retrieved by the Cache Fetcher. The Cache Fetcher also encodes the data using the Encoder.
- The Window Buffer Manager and Cache Manager are responsible for managing the input data from the RDF streams and RDF datasets, respectively; and for feeding them into the Dynamic Executor.
- The Dynamic Executor enables a dynamic execution strategy where the query plan can be changed during the life time of a continuous query. The most efficient query plan is continuously advised by the Adaptive Optimizer based on data statistics and operator costs.
- The outputs of the Dynamic Executor have to be decoded by the Decoder before being streamed out. The Encoder and Decoder share a Dictionary for encoding and decoding.

CQELS Engines: Embedded to Cloud

To be a platform-agnostic execution framework, the design of the aforementioned components of the CQELS Framework is abstract enough to be realized by various hosting platforms which can provide different library/software stacks as well as different hardware architectures. The first version of CQELS engine [10] was implemented as a stand-alone version for PCs. It uses popular libraries like Apache Jena [13], high performance computing data structures [8] for implementing new data structures for Window Buffer Manager associated with in-memory incremental evaluation algorithms for sliding window operators [9]. This version is accompanied with several adaptive optimisation algorithms to boost the processing throughput of CQELS engine.

To build CQELS engines for resource constrained environments like embedded devices, mobile phones, we build embedded CQELS engine by reducing code footprint. We use light-weight data structures for Window Buffer as well as reuse RDF On The Go (RDF-OTG) [12] code based for compact versions of Encoder/Decoder, Dictionary, Cache Manager/Fetcher. Besides, the CQELS embedded engine extends SPARQL physical query operators of RDF-OTG to sliding windows operators. The whole embedded CQELS is smaller than 10MB and needs only 4-6MB of RAM to process millions of triples on various small devices such as BeagleBone [1], Intel Galileo [3], Raspberry PI [4], and any Android Phones.

For scalability, we use Storm [5] and HBase [2] as underlying software stacks for coordinating parallel execution processes to build an RSP engine on the cloud computing infrastructure, called CQELS Cloud [11]. We adapt highly efficient algorithms of CQELS engine on stand-alone PCs to the distributed share-nothing

architecture, thus, CQELS Cloud can scale up to million inputs per second with 100.000 concurrent queries on a cluster of 32 EC2 computing nodes.

Development Roadmap

Thanks to the RSP working group, the community has brought various interesting use cases that create new requirements to build more features for our future CQELS engines. The new feature to be implemented in the next release is supporting event processing pattern with basic reasoning rules such as temporal, spatial and RDFS. In the long term, full pledge stream reasoning will be enabled along the line with state of the art of emerging research effort on this area [14]. The rise of Internet of Things inspires us to build much smaller CQELS engine to be able to run on tiny micro controller, sensor boards which have less than 50KB of RAM. Moreover, instead of having to embed to the application, the client/server mechanism will be supported using Websockets and MQTT. Last but not least, adapting agreed query syntax and data model from the RSP working group is a step forward to support the standardised CQELS-QL.

Acknowledgment

This publication has emanated from research supported in part by research grants from Irish Research Council under Grants No. GOIPD/2013/104 and No. GOIPG/2014/917, European Commission under Grant No. FP7-ICT-608662 (VITAL), and by the Austrian Science Fund (FWF) project P26471.

References

1. Beaglebone. `http://beagleboard.org/`.
2. Hbase. `http://hbase.apache.org/`.
3. Intel Galileo. `http://www.intel.com/content/www/us/en/do-it-yourself/galileo-maker-quark-board.html`.
4. Raspberry. `http://www.raspberrypi.org/`.
5. Storm. `https://storm.apache.org/`.
6. Dave Beckett and Jeen Broekstra. SPARQL Query Results XML Format (Second Edition). `http://www.w3.org/TR/rdf-sparql-XMLres/`, 2013. [Online; accessed 15-March-2015].
7. Steve Harris and Andy Seaborne. SPARQL 1.1 Query Language. `http://www.w3.org/TR/sparql11-query/`, 2013. [Online; accessed 15-March-2015].
8. Steve Harris and Andy Seaborne. Fast and compact type-specific collections for java. `http://fastutil.di.unimi.it/`, 2015. [Online; accessed 15-March-2015].
9. Danh Le Phuoc. *A Native And Adaptive Approach for Linked Stream Processing*. PhD thesis, National University of Ireland, Galway, 2013.
10. Danh Le-Phuoc, Minh Dao-Tran, Josiane Xavier Parreira, and Manfred Hauswirth. A native and adaptive approach for unified processing of linked streams and linked data. In *ISWC 2011 (1)*, pages 370–388, 2011.

11. Danh Le Phuoc, Hoan Nguyen Mau Quoc, Chan Le Van, and Manfred Hauswirth. Elastic and scalable processing of linked stream data in the cloud. In *ISWC 2013 (1)*, pages 280–297, 2013.
12. Danh Le Phuoc, Anh Lê Tuán, Gregor Schiele, and Manfred Hauswirth. Querying heterogeneous personal information on the go. In *ISWC 2014 (2)*, pages 454–469, 2014.
13. Andy Seaborne. Apache jena: A free and open source java framework for building semantic web and linked data applications. `https://jena.apache.org/`, 2015. [Online; accessed 15-March-2015].
14. Emanuele Della Valle, Stefano Ceri, Frank van Harmelen, and Dieter Fensel. It's a streaming world! reasoning upon rapidly changing information. *IEEE Intelligent Systems*, 24(6):83–89, 2009.