RDF Stream Reasoning via Answer Set Programming on Modern Big Data

Xiangnan Ren, Olivier Curé, Hubert Naacke, Guohui Xiao

RDF Stream Reasoning via Answer Set Programming on Modern Big Data Platform

Xiangnan Ren[1,2], Olivier Curé[2], Hubert Naacke[3], and Guohui Xiao[4]

[1] ATOS, France
xiang-nan.ren@atos.net
[2] LIGM (UMR 8049), CNRS, UPEM, France
olivier.cure@u-pem.fr
[3] Sorbonne Universités, UPMC, Univ Paris 06, France
hubert.naacke@lip6.fr
[4] Free University of Bozen-Bolzano, Italy
xiao@inf.unibz.it

Abstract. RDF stream reasoning is gaining more and more attention but current research mainly focuses on logical frameworks which aim to formalize the query semantics and enhance the complexity of reasoning ability. These frameworks are evaluated on prototype systems based on a centralized design and suffer from limited scalability. A common way to enhance system scalability is to adopt a distributed approach. Moreover, the study of applying distributed solution for expressive RDF stream reasoning is still missing. In this paper, we explore the ability of modern Big Data platform to handle highly expressive temporal Datalog/Answer Set Programming(ASP) over RDF data streams. In order to achieve our goal, we first discuss some key features to parallelize Datalog/ASP program, and we associate these features to the two well known distributed stream processing models, namely Bulk Synchronous Processing (BSP) and Record-at-A-Time (RAT). We build a technical demonstrator called BigSR on top of Spark(BSP) and Flink(RAT) to support our evaluations, and identify the pros and cons of each model. Our experiments show that, BigSR achieves high throughput beyond million-triples per second using a rather small cluster of machines.

1 Introduction

In the era of the ever-growing semantic data flood, the challenge of processing declarative queries and inferences over rich and massive RDF data streams remains of major issue. On the one hand, stream processing must be efficient enough to ingest data with throughput and latency constraints which are imposed respectively by the incoming data streams and underlying applications. On the other hand, the query language has to be expressive enough to support temporal logic and reasoning that may require recursion. In order to cope with the first aspect, distributed systems supporting fault tolerance, automatic task distribution and recovery are generally required. Considering the second aspect, Datalog [1] and Answer Set Programming (ASP)[4] programs seem to fit efficiently since they represent a good balance between expressive power, safety, performance, and usability. Note that considering such expressiveness permits to address ontology languages such as OWL2RL. This work demonstrates the feasibility to design such a system but it also emphasizes that such a solution can be implemented with open-source, state of the art Big Data technologies, hence being a prototype for a production-ready system.

2 Distributed RDF Stream Reasoning

We use LARS [2] as the theoretical foundation of our implementation. In addition, we identify **Parallelism Level** and **Streaming Model** as the two main factors which leverages the scalability of distributed RDF stream reasoning.

Parallelism Level. As defined in [5], there are three levels to parallelize the evaluation of a stratified Datalog/ASP program: *Component Level*, *Rules level*, and *Single Rule level*. Through our work, we designed a series of queries which cover all the three parallelism level to evaluate the performance impacts.

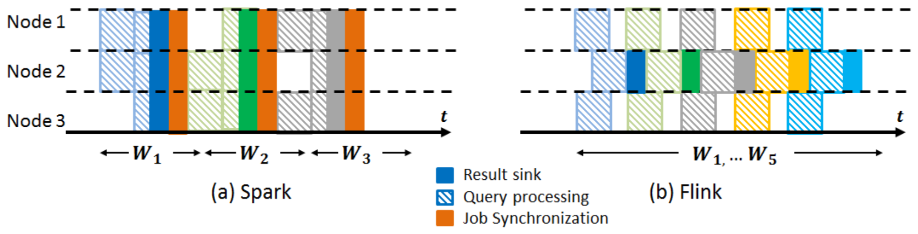


Fig. 1: Blocking and non-blocking query processing.

Streaming Models. Two broad classes of streaming models are adopted by modern distributed streaming engines, *i.e.*, Bulk Synchronous Processing (BSP) and Record-at-A-Time (RAT). In order to evaluate the efficiency of these two streaming models for RDF stream reasoning, we decide to choose Apache Spark Streaming (*i.e.*, of BSP) [6] and Apache Flink (*i.e.*, of RAT) [3] as the underlying computing frameworks. Considering a simple LARS program $\mathcal{P} = \mathrm{T}(X) \leftarrow \boxplus_{\tau}^{w(l,d)} \Diamond(\mathrm{R1}(Y) \wedge \mathrm{R2}(Y, X))$, Figure 1 compares the differences of program evaluations between BSP and RAT on Spark and Flink, respectively. Spark launches the continuous query execution synchronously, each query execution is triggered after the previous computations is completed (Figure 1(a)). Flink serializes, caches, and pushes forward each record to the next operator eagerly right after the current computation is done. The asynchronous data processing on Flink minimizes processing delay (Figure 1(b)).

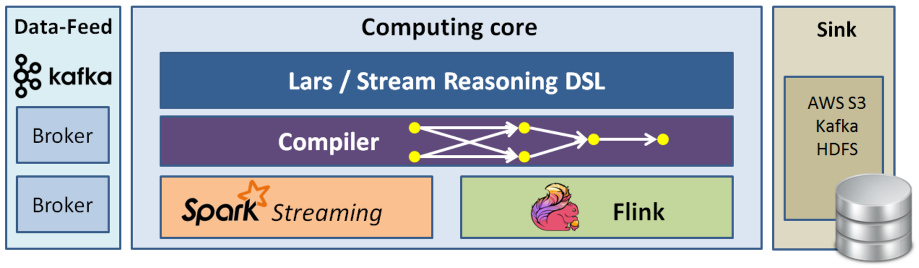


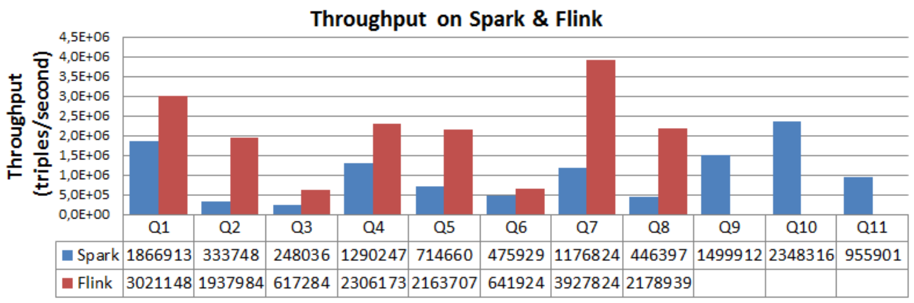
Fig. 2: BigSR system architecture

BigSR. To study the feasibility of applying a distributed approach on RDF stream reasoning and explore the performance impact associated with two above-mentioned factors, we build BigSR - a reusable prototype for distributed RDF stream reasoning. BigSR consists of three principal components: (i) *Data-feed* is built on top of Apache Kafka (a distributed message queue) for high throughput, fault-tolerant data stream management; (ii) *Sink* persists query outputs into a storage component such as Amazon S3, HDFS or even Kafka; (iii) *Computing core* compiles LARS program into BigSR's logical plan and evaluates the program via Spark/Flink's native operators.

3 Evaluation

In this section, we showcase some evaluation result. Following the standard Yahoo benchmark tailored for streaming systems, we designed a micro-benchmark to evaluate BigSR. The benchmark involves 15 queries and 4 datasets (SRBench, CityBench, Lubm and Waves). We organize the 15 queries into two groups: (1) in the first group, the queries Q_1 to Q_{11} are designed to evaluate the two main factors of streaming engine, *i.e.*, system throughput and query latency. In particular, we add recursive operators in Q_9, Q_{10} and Q_{11} to study the pros and cons of recursion support on BSP and RAT. (2) the queries Q_{12} to Q_{15} in the second group are designed for the purpose of evaluating the minimum latency that the system could achieve.

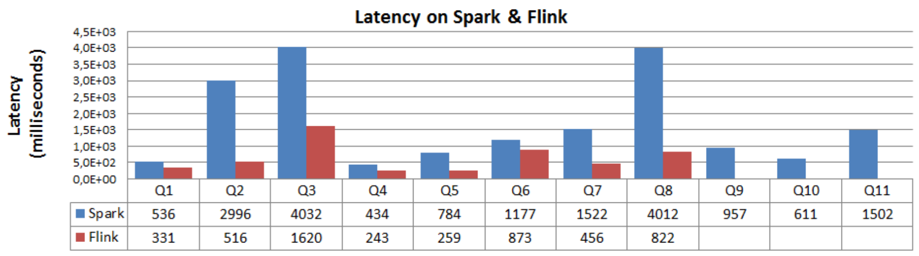
We evaluate BigSR [1] in a small cluster of 9 nodes (6 nodes for Spark/Flink, 3 nodes for Kafka and ZooKeeper). Figure 4 and Figure 4 respectively give the engine throughput and query latency of Q_1 to Q_{11}. On both Spark and Flink, BigSR attains throughput of millions triples per second, and second-level delay. Table 1 compares the query latency of Q_{12} to Q_{15}. Since Spark requires to define the size of micro-batch within its BSP streaming model, we reduce the size of micro-batch to 500 ms and record the processing delay. In general, limited by the BSP model, Spark retains the latency around 100 ms. In the contrary, RAT model provides Flink the ability to achieve the latency of sub-millisecond.



	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11
🟦 Spark	1866913	333748	248036	1290247	714660	475929	1176824	446397	1499912	2348316	955901
🟥 Flink	3021148	1937984	617284	2306173	2163707	641924	3927824	2178939			

Fig. 3: Throughput (milliseconds) on Spark and Flink for Q_1 to Q_{11}.

[1] https://github.com/renxiangnan/bigsr



	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11
🟦 Spark	536	2996	4032	434	784	1177	1522	4012	957	611	1502
🟥 Flink	331	516	1620	243	259	873	456	822			

Fig. 4: Query latency (milliseconds) on Spark and Flink for Q_1 to Q_{11}.

	Q_{12}	Q_{13}	Q_{14}	Q_{15}
Spark	110	96	115	99
Flink	<1	<1	<1	<1

Table 1: Stateless query latency (millisecond); Spark micro-batch size = 500 ms.

4 Conclusion

Expressive RDF stream reasoning is an emerging area that is in its infancy. The research for scalable RDF stream reasoning, especially by applying distributed approach, is still missing. In this paper, we introduce BigSR and some results of experiments. We use LARS as the theoretical foundations for our implementations, and we build a bridge between recent stream reasoning theoretical work and modern Big Data technology. Our evaluation shows that distributed solution enhance the system throughput to million triples per second with second/sub-second delay. In future work, we plan to concentrate on the trade-off between the query expressiveness and system scalability, which gives us a road map to design a production-ready engine.

Acknowledgements. This work was partially supported by the OBATS project at the Free University of Bozen-Bolzano.

References

1. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. Harald Beck, Minh Dao-Tran, Thomas Eiter, and Michael Fink. LARS: A logic-based framework for analyzing reasoning over streams. In *AAAI*, 2015.
3. Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache flink$^{\mathrm{TM}}$: Stream and batch processing in a single engine. *IEEE Data Eng. Bull.*, 2015.
4. Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. Answer set programming: A primer. In *Reasoning Web*, 2009.
5. Simona Perri, Francesco Ricca, and Marco Sirianni. Parallel instantiation of ASP programs: techniques and experiments. *TPLP*, 2013.
6. Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *NSDI*, 2012.