StreamRule: A Nonmonotonic Stream Reasoning System for the Semantic Web

Alessandra Mileo, Ahmed Abdelrahman,
Sean Policarpio, and Manfred Hauswirth

DERI, National University of Ireland, Galway, Ireland
`first.last@deri.org`

Abstract. Stream reasoning is an emerging research field focused on dynamic processing and continuous reasoning over huge volumes of streaming data. Finding the right trade-off between scalability and expressivity is a key challenge in this area. In this paper, we want to provide a baseline for exploring the applicability of complex reasoning to the Web of Data based on a solution that combines results and approaches from database research, stream processing, and nonmonotonic logic programming.

1 Introduction and Motivation

The Semantic Web and the growing interests in linking data for sharing, re-use, and understanding have started to intersect with the domain of *Big Data*[1]. To be successful and efficient in this joint space, processing paradigms need a shift from the current batch-like approaches (e.g., distributed and parallel computing with MapReduce) towards stream reasoning in near-real-time [10].

Stream processing is under active research for several years in Database as well as in the Semantic Web community [8,21,20,7]. *Stream reasoning* on the other hand had only started in recent years, and a few research prototypes exist [9,17,18,4]. Some of these solutions try to capitalize on the synergies between stream query processing and stream reasoning for Web data, but the trade-off between scalability and expressivity is not exploited enough yet. Gaining better insights on this trade-off is a necessary step to be able to deal with large volume of input streams, incomplete or inconsistent data, and computationally intensive reasoning tasks. Our approach for this investigation combines the latest advancements in stream query processing for linked data on the Web, with nonmonotonic stream reasoning. To our knowledge, ours is the first approach that proposes this kind of coupling to explore how to enable expressive reasoning for the Semantic Web.

Nonmonotonic stream reasoning techniques for the (Semantic) Web have potential impact in a variety of real-world applications, but the ability of dealing with incomplete and noisy input streams, conflicts, defaults, qualitative preferences, constraints, non-determinism and generation of multiple solutions is computationally intensive. Extensions of Datalog towards the logic paradigm of Answer Set Programming (ASP) [16,5] have been implementing these reasoning

[1] `http://semanticweb.com/big-data-and-the-semantic-web-their-paths-will-cross_b32027`

capabilities which can go far beyond the capabilities of existing query engines. Logic programming dialects like Datalog with negation, covered by ASP, are viewed as a natural basis for the Semantic Web rule layer [13], but the full expressivity of ASP introduces new challenges concerning the trade-off between expressivity and scalability, especially in a streaming scenario.

Our approach is based on the assumption that not all raw data from the input stream might be relevant for complex reasoning, and efficient stream query processing can provide aggregation and filtering functionalities to help reducing the information load over the logic-based stream reasoner.

In order to deal with streaming RDF, solutions like the *Linked Sensor Middleware* (LSM) [22] have been proposed, providing APIs and a SPARQL-like endpoint abstraction, CQELS [20], for efficiently [19] processing SPARQL queries on streaming and static RDF data. In terms of knowledge-intensive stream reasoning a recent ASP-based solution [15], shifts the emphasis from rapid data processing towards complex reasoning needed in domain such as robotics, planning, health monitoring and so on. The resulting system *OClingo* [15], has been proven to outperform other nonmonotonic reasoners [1], but its applicability to the Semantic Web requires tighter coupling with efficient stream processing techniques. For this reason, we believe the combination of LSM and CQELS for stream filtering and aggregation, with OClingo represents a good starting point for investigating the applicability of complex reasoning for the Semantic Web.

To demonstrate this idea, we consider the scenario below.

Scenario 1.1 *People wear RFID tags and move around in a building equipped with RFID readers producing streams of position information. Within the building, we have defined "geo-fences", i.e., virtual perimeter for a real-world area. Particular rooms are marked "off-limits." We want to detect (1) users who have traversed outside the geo-fence or into off-limits rooms, and (2) inconsistencies in the movement patters of people, e.g., movement between non-adjacent rooms.*

Each RFID tag reading generates an event (a person enters a room) captured by the stream query processor. In the reasoning step, we could detect a geo-fence violation under incomplete knowledge by reasoning about the *absence* of knowledgeor by triggering default behaviours. For example, if a subject *Bob* cannot be identified or we don't know if he has access to a room X, we can still detect a geo-fence violation with a rule like: *"If **somebody** is detected in Room X but we **do not know** whether his access is restricted, then send an alert"*. This is generalized to all persons regardless of our ability to identify them or prior knowledge about their restricted access. In a similar way, if we have no sensor readings available, we may want to know all plausible locations for a person in a particular point in time, generating plausible solutions or "answer sets".

Nonmonotonicity via the use of Negation As Failure (NAF) and non determinism makes it possible to address these cases. NAF can also be applied in reasoning about inconsistencies, such as movements between unconnected rooms (e.g. rooms with no path between them or connected by locked doors). To do that, we can define a rule like *"If Bob is detected in both room X and room Y in the last 5 seconds, and room X and room Y are **not** connected, record an inconsistency"*. Similar to our previous rules, we reason upon the streaming information regardless of the possible absence of some knowledge (in this case, a connection between rooms X and Y). Additionally, we specify temporal reasoning by constraining the rule to a specific window (i.e., the last 5 steps).

Non determinism can be combined with special constraints to produce alternative solutions. For example, if *"Bob is in room X and in room Y"* because of erroneous readings, rather than reporting this as an inconsistency we can define a constraint rule saying that *"For all of Bob's sensor readings, only one of them only can be valid in each solution"*. In ASP this translates in each stream reading for *Bob* to be part of only one of the solutions. If this information related to some other rules – for example, setting off an alert – then they will be triggered in their respective answer set too.

2 System Design and Implementation

The conceptual design of *StreamRule* is shown in Figure 1. Abstraction and filtering on raw streaming data is performed by a *stream query processor* using declarative query patterns as filters. Matching patterns are processed by the middle layer and returned as input facts to the nonmonotonic stream reasoner together with the declarative encoding of the problem at hand. Output is returned as a set of solutions to the logic program applied to the (filtered) input.

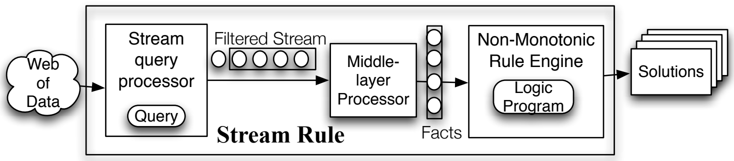

Fig. 1. StreamRule: Conceptual Framework

Event filtering is specified by end-users through a *CQELS* query[2]. As the *CQELS* engine provides API accessibility for configuration, instantiation, and output using Java, we also based the *StreamRule* implementation on Java. Our system acts as a pipeline which intercepts the output RDF stream query results filtered by *CQELS* and processes them into internal data structures. These are then streamed into *OClingo* in ASP syntax. Presently, *OClingo* is executed as a subprocess within *StreamRule* and the answer sets it produces are fed back into the Java runtime for further processing and/or display. To optimize performance and guarantee non-blocking communication and processing, *StreamRule* uses asynchronous communication and a multi-threaded and concurrent callback-based (or *chaining*) infrastructure across and among the three major components – *CQELS*, *OClingo*, and the Java middle layer. We also implemented a *buffering* algorithm to ensure the reasoner does not become a bottleneck. Additionally, the implementation also adheres to accepted best practices for performance-optimized stream

[2] SPARQL 1.1 extension with streaming operators.

processing engine design [23]. The configuration of the *StreamRule* system is currently specified as an an XML configuration file. The base system is then capable of being run directly in a Java Virtual Machine. After deployment and configuration into a web server, users are simply required to define two major inputs: the RDF stream query and the logic program. Once execution is started, continuous answer sets are output to a dynamic user interface. The reasoning component uses *OClingo*'s sliding windows and time-decay model [15].

3 Related Work

Stream reasoning and processing is a rapidly growing area of research. Relevant systems include *C-SPARQL* [6], *EP-SPARQL* [2]. A recent effort is *Sparkwave* [17] which, however, does not have comprehensive performance evaluation results available, thus making comparison hard.

Due to the decisive results in [19], our preference was towards *CQELS*. In general though, these systems share the same approach of utilizing SPARQL-like specification of continuously processed queries for streaming RDF data, so they could all be used for *StreamRule* in principle. What they lack is the possibility to express more complex reasoning. Stream processing engines which augment stream reasoning through this kind of approach are still limited, but include those such as the use of *Prova* [24] and *Streaming IRIS* [3]. In terms of performance, *Prova* is more concerned about how much background (static) knowledge can be pushed into the system, while *Streaming IRIS* does not test complex reasoning tasks. *ETALIS* [3] is a CEP system implemented in Prolog, which provides a rich temporal query language to capture some of the patterns we considered. To our knowledge, the work by Do et al. [12] is probably the only other current stream reasoning approach for the Semantic Web that utilizes ASP. Although the work is quite recent, their approach is still much more prototypical and it is based on the *DLV* [14] system, which does not pertain to continuous and window-based reasoning over streaming data within the reasoner.

4 Experiments and Discussion

To evaluate our implementation and trigger discussions we enacted Scenario 1.1 using real sensor readings [4]. The CQELS queries and the ASP rules are available at `http://streamrule.org/www13/experiments`. Assuming the expressive power of ASP is clear to the reader, in what follows we briefly discuss our lesson learned as a starting point for future investigation.

Streaming rate and program complexity: We observed that the logic program needs to return results faster than the inputs arrive from the stream query component. While this may look like a limitation, it provides a clear understanding of the throughput threshold up to which the system is guaranteed to work correctly, and can be used for planning the input stream

[3] `http://www.envision-project.eu/wp-content/uploads/2012/11/D4.8-1.0.pdf`

[4] `http://www.openbeacon.org/`

capacities and the necessary hardware for the reasoning. Such hard boundaries are also used in real-time systems to ensure upper bounds in the result delays of a computation. We follow this established technique in *StreamRule*.
Streaming rate and window size: A trade-off exists between logical window size and streaming rate: faster streams are more likely to produce query matches, so they require smaller window sizes, unless the speed of the reasoning process is increased by faster hardware. Conversely, if larger window sizes are required, then sampling techniques or further aggregation or selective reading might help. This is reflected in most real scenarios where we need to decide what is a relevant event pattern for a particular reasoning task and for how long that is likely to be valid. Leveraging fast stream data with the use of *CQELS* query patterns as a filter, can provide significantly lower requirements for the computation thus enabling some forms of complex reasoning, but further investigation for optimization is needed.
Optimizations: Our initial investigation helped understanding limitations and feasibility of applying complex reasoning over noisy and erroneous linked data streams, capacities that make *StreamRule* a stream reasoning system in a class of its own [11]. This work also help sketching a promising path for optimization of such system which requires i) definition of an operational semantics for tighter integration of the two components, ii) design and investigation of more complex scenarios to better understand the nature of the correlation between window-size and streaming rate and iii) specification of dynamic metrics for the reasoner (e.g. based on the complexity of the ruler or the relevance and quality of the streaming events) to provide a feedback loop to the query processor. In this way event filtering and optimization can be tailored to the specific scenario and data at hand. We are currently investigating these issues and engaging with scientists from relevant domains including smart cities and bioinformatics.

References

1. 2011 ASP Competition – system track final results (2011), `https://www.mat.unical.it/aspcomp2011/SystemTrackFinalResults`
2. Anicic, D., Fodor, P., Rudolph, S., Stojanovic, N.: EP-SPARQL: a unified language for event processing and stream reasoning. In: Proc. of the 20th WWW Conference, pp. 635–644. ACM (2011)
3. Anicic, D., Fodor, P., Rudolph, S., Stühmer, R., Stojanovic, N., Studer, R.: A rule-based language for complex event processing and reasoning. In: Hitzler, P., Lukasiewicz, T. (eds.) RR 2010. LNCS, vol. 6333, pp. 42–57. Springer, Heidelberg (2010)
4. Anicic, D., Rudolph, S., Fodor, P., Stojanovic, N.: Stream reasoning and complex event processing in ETALIS. In: Semantic Web (2011)
5. Baral, C.: Knowledge representation, reasoning and declarative problem solving. Cambridge University Press (2003)
6. Barbieri, D., Braga, D., Ceri, S., Grossniklaus, M.: An execution environment for c-sparql queries. In: Proc. of the 13th Int'l Conference on Extending Database Technology, pp. 441–452. ACM (2010)

7. Barbieri, D.F., Braga, D., Ceri, S., Della Valle, E., Grossniklaus, M.: Querying rdf streams with c-sparql. SIGMOD Record 39(1), 20–26 (2010)
8. Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Seidman, G., Stonebraker, M., Tatbul, N., Zdonik, S.: Monitoring streams: a new class of data management applications. In: VLDB 2002, pp. 215–226. VLDB Endowment (2002)
9. Della Valle, E., Ceri, S., Barbieri, D.F., Braga, D., Campi, A.: A first step towards stream reasoning. In: Domingue, J., Fensel, D., Traverso, P. (eds.) FIS 2008. LNCS, vol. 5468, pp. 72–81. Springer, Heidelberg (2009)
10. Della Valle, E., Ceri, S., van Harmelen, F., Fensel, D.: It's a Streaming World! Reasoning upon Rapidly Changing Information. IEEE Intelligent Systems 24(6), 83–89 (2009)
11. Della Valle, E., Schlobach, S., Krötzsch, M., Bozzon, A., Ceri, S., Horrocks, I.: Order matters! harnessing a world of orderings for reasoning over massive data. Semantic Web Journal (2012)
12. Do, T.M., Loke, S.W., Liu, F.: Answer set programming for stream reasoning. In: Butz, C., Lingras, P. (eds.) Canadian AI 2011. LNCS (LNAI), vol. 6657, pp. 104–109. Springer, Heidelberg (2011)
13. Eiter, T., Ianni, G., Polleres, A., Schindlauer, R., Tompits, H.: Reasoning with rules and ontologies. In: Barahona, P., Bry, F., Franconi, E., Henze, N., Sattler, U. (eds.) Reasoning Web 2006. LNCS, vol. 4126, pp. 93–127. Springer, Heidelberg (2006)
14. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: Dlv-hex: Dealing with semantic web under answer-set programming. In: Proc. of ISWC (2005)
15. Gebser, M., Grote, T., Kaminski, R., Obermeier, P., Sabuncu, O., Schaub, T.: Answer set programming for stream reasoning. CoRR, abs/1301.1392 (2013)
16. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Proc. of the 5th Int'l Conference on Logic Programming, vol. 161 (1988)
17. Komazec, S., Cerri, D., Fensel, D.: Sparkwave: continuous schema-enhanced pattern matching over RDF data streams. In: Proc. of DEBS, pp. 58–68 (2012)
18. Lanzanasto, N., Komazec, S., Toma, I.: Reasoning over real time data streams (2012), http://www.envision-project.eu/wp-content/uploads/2012/11/D4-8_v1-0.pdf
19. Le-Phuoc, D., Dao-Tran, M., Pham, M.-D., Boncz, P., Eiter, T., Fink, M.: Linked stream data processing engines: Facts and figures. In: Cudré-Mauroux, P., et al. (eds.) ISWC 2012, Part II. LNCS, vol. 7650, pp. 300–312. Springer, Heidelberg (2012)
20. Le-Phuoc, D., Dao-Tran, M., Xavier Parreira, J., Hauswirth, M.: A native and adaptive approach for unified processing of linked streams and linked data. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 370–388. Springer, Heidelberg (2011)
21. Madden, S., Shah, M., Hellerstein, J.M., Raman, V.: Continuously adaptive continuous queries over streams. In: 2002 ACM SIGMOD Int'l Conference on Management of Data, pp. 49–60. ACM, New York (2002)
22. Le-Phuoc, D., Quoc, H.N.M., Xavier Parreira, J., Hauswirth, M.: The Linked Sensor Middleware – Connecting the real world and the Semantic Web. In: Semantic Web Challenge, ISWC, Bonn, Germany, October 23-27 (2011)
23. Stonebraker, M., Çetintemel, U., Zdonik, S.: The 8 requirements of real-time stream processing. ACM SIGMOD Record 34(4), 42–47 (2005)
24. Teymourian, K., Rohde, M., Paschke, A.: Fusion of background knowledge and streams of events. In: Proc. of the ACM DEBS, DEBS 2012, pp. 302–313. ACM, New York (2012)