Towards Smart Cache Management for Ontology Based, History-Aware Stream Reasoning

Rui Yan[1], Brenda Praggastis[2], William P. Smith[2], and Deborah L. McGuinness[1]

[1] Tetherless World Constellation, Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY, USA
yanr2@rpi.edu,dlm@cs.rpi.edu ,
[2] Pacific Northwest National Laboratory, Richland, WA, USA
{Brenda.Praggastis,William.Smith}@pnnl.gov

Abstract. Stream reasoning is an exciting multidisciplinary research area that combines stream processing and semantic reasoning. Its goal is to not only process a dynamic data stream, but also to extract explicit and implicit information on-the-fly. One of its challenges is managing history awareness: how much and which historical data should be held and for how long as we continuously query and reason on an ever changing stream of linked data? In this paper, we propose an innovative approach to enable history-aware reasoning by utilizing semantic technologies in a data cache with a statistics-based cache management policy.

Keywords: semantic web, stream reasoning, cache management

1 Introduction

Examples of streaming data are ubiquitous on the web. Data streams are presented in a multitude of physical formats and conceptual models making analysis difficult. Inspired by the Linked Open Data approach to link static heterogeneous data on the Web, D. Barbieri et al[3] launched an initiative to publish data streams as linked data modeled using the Resource Description Framework (RDF).

Existing (semantic) stream processing techniques[4][7] are capable of quickly processing large volume RDF streams, yet typically perform superficial manipulations of information. Reasoning supported by semantic technologies can infer meaning, but is mostly performed against a static knowledge base. The rising need to extract high level knowledge from streaming data gave birth to stream reasoning[6], leveraging the advantages of both stream processing and semantic reasoning. Several use cases of stream reasoning are described in[8], and from these cases general requirements for effective stream reasoning are proposed. Historical data management is among these requirements.

Most existing solutions of stream processing and reasoning use a window to deal with the transient nature of streaming data. The window can be either time-based or count-based to isolate the most recent segment of the unbounded

RDF streaming data. Both C-SPARQL[4] and EP-SPARQL[1] extend the standard SPARQL to enable a continuous query within the window. The difference between them is the former deals with RDF data streams, while the latter processes complex events. The algorithm of incrementally maintaining ontological entailments[5] employs time-interval-stamped RDF streams and C-SPARQL to constantly maintain the transitive property entailments as expired RDF streams are dropped.

A window slides along the RDF data stream in the order the data appear, inserting and evicting data according to this order. Decisions of which data will be evicted in a window-based stream reasoning system are made a priori. Historical data, data which have slid out of the window, are lost regardless of their semantic importance. In contrast, our approach will decide which data to remove using semantic importance criteria. A cache holds a portion of the data to be processed as the data are fed in, maintaining a history of semantically important data by evicting data according to a cache management algorithm. For example, the Least Frequently Used (LFU) algorithm counts data reasoning participation frequency and evicts data with small frequency counts; while the Least Recently Used (LRU) algorithm evicts data with less recent reasoning involvements. Caching management algorithms have been well-studied and applied on the Web over decades[2]. In this paper, we propose an innovative approach to enable history-aware reasoning by utilizing semantic technologies in a data cache with a statistics-based cache management policy.

2 Approach

Our approach is self-adaptive in that it manages the historical data by constantly updating cached data with a cache management algorithm. We present this approach based on our Nuclear Magnetic Resonance (NMR) use case[9], which exploits stream reasoning to identify chemical compound types in test specimens. We model the background ontology in the OWL 2 DL profile, describing thirty different compound categories with their resonance spectra value ranges. The NMR streaming data contain compound instances with their resonance spectra frequency values, formatted in Avro[1] and then annotated into RDF graphs. The goal of this use case is to identify categories of streaming instances via DL reasoning performed by Pellet[2]. A compound instance is identified in a category if its frequency value fits into an ontology-encoded category range. The issue we are addressing here is the time-consuming reasoning. For example, in our no-cache experiment it took 19 minutes to produce entailments for ∼19,000 triples in the stream, which violated the timely nature of stream reasoning .

The data cache, whose size is fixed due to the maintenance feasibility and reasoning limitations, is a key component for our stream reasoning architecture. The cache is preloaded with the background ontology and equipped with a statistics-based cache management algorithm. In general, the choice of these

[1] https://avro.apache.org/
[2] https://github.com/complexible/pellet

algorithms is dependent on the system requirements; in the NMR use case, our algorithm will be based on LFU.[3] When an instance is identified as one of the desired compounds, it becomes important and its original and entailed graph is saved in the data cache as a historical data graph. We associate a counter for each historical data graph, and use a use case defined predicate NMR:frequency to pull out frequency values of both streaming and historical data graphs. Identical instances are identified by comparing their frequency equality with SPARQL's numeric-equal operator. The counter increments by one if the operator returns true. Otherwise, the streaming instance has to go through the reasoner. The bigger the counter, the more semantically important the historical data graph. A historical data graph can either become more semantically important (counter grows bigger) to stay or less semantically important (counter grows smaller) to evict as new data are fed in. Using the historical data, new compounds with identical frequency values will be identified without having to go through the full reasoning process, which saves time.

We will implement this approach in multi-threading: one thread for a continuous standard SPARQL query, the other one for cache management. These two threads are executed in parallel and can access the data in the cache. The standard SPARQL thread queries the triples held in the cache, returning a list of identified compounds. Continuity is enabled by repeating the query at regular time intervals. The cache management thread increments the counters and repeatedly checks the counter values, so as to evict certain compound graphs if they are not frequently used.

3 Discussion

Introducing a data cache into the stream reasoning system is beneficial because of its ability to maintain historical data; this not only provides extra background for reasoning, but also preserves important history for potential use, such as trend identification, and anomaly detection. For example, in our NMR use case, a large statistical count of the historical compound instance data indicates a greater likelihood that the test specimen contains a large percentage of this compound. By identifying identical data in historical and streaming data we decrease overhead of reasoning, as illustrated above. The cache is also suitable to work with existing window-based semantic stream processing engines such as C-SPARQL. The data cache can replace the window in such engines for stream reasoning under those scenarios emphasizing on historical data and fast system response.

Multi-threading is a beneficial way to boost system performance because it splits different tasks into threads and executes them in parallel. Assigning one thread to repeatedly evaluate a standard SPARQL query realizes an easy but effective continuous query. This is different than proposals like C-SPARQL, which introduce extra syntax and a different execution model for query evaluation.

[3] Different algorithms' performances and effects on our stream reasoning systems will be left as future work.

Last but not least, the cache management algorithm needs to be efficient to avoid data congestion by spending as little time as possible on deciding data removal.

4 Conclusion and Future Work

To conclude, we have proposed an innovative approach to enable history-aware reasoning by utilizing semantic technologies in a data cache with a cache management algorithm. As for our future work, we will explore several effective cache management algorithms, impacts of ontology expressiveness on the system performance, as well as evaluation methods to benchmark our approach.

5 Acknowledgments

The research described in the paper is part of the AIM Initiative at PNNL. It was conducted under the Laboratory Directed Research and Development (LDRD) program at PNNL, a multi-program national laboratory operated by Battelle for the U.S. Department of Energy under contract DE-AC06-76RLO 1830.

References

1. Darko Anicic, Paul Fodor, Sebastian Rudolph, and Nenad Stojanovic. Ep-sparql: a unified language for event processing and stream reasoning. In *Proceedings of the 20th international conference on World wide web*, pages 635–644. ACM, 2011.
2. Abdullah Balamash and Marwan Krunz. An overview of web caching replacement algorithms. *Communications Surveys & Tutorials, IEEE*, 6(2):44–56, 2004.
3. Davide F Barbieri and ED Valle. A proposal for publishing data streams as linked data. In *Linked Data on the Web Workshop*, 2010.
4. Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. C-sparql: Sparql for continuous querying. In *Proceedings of the 18th international conference on World wide web*, pages 1061–1062. ACM, 2009.
5. Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. *Incremental reasoning on streams and rich background knowledge*. Springer, 2010.
6. Emanuele Della Valle, Stefano Ceri, Davide Francesco Barbieri, Daniele Braga, and Alessandro Campi. *A first step towards stream reasoning*. Springer, 2009.
7. Marcin Gorawski, Anna Gorawska, and Krzysztof Pasterak. A survey of data stream processing tools. In *Information Sciences and Systems 2014*, pages 295–303. Springer, 2014.
8. Alessandro Margara, Jacopo Urbani, Frank van Harmelen, and Henri Bal. Streaming the web: Reasoning over dynamic data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 25:24–44, 2014.
9. Smith WP and MI Borkum. Semantically enabling stream-reasoning architectural frameworks. Abstract submitted to The 24th ACM International Conference on Information and Knowledge Management (CIKM 2015), Melbourne, Australia., 2015. PNNL-SA-110228.