Towards A Cache-Enabled, Order-Aware, Ontology-Based Stream Reasoning Framework

Rui Yan
Tetherless World Constellation, Department of Computer Science, Rensselaer Polytechnic Institute
Troy, NY, USA
yanr2@rpi.edu

Brenda Praggastis
Pacific Northwest National Laboratory
Richland, WA, USA
Brenda.Praggastis@pnnl.gov

William P. Smith
Pacific Northwest National Laboratory
Richland, WA, USA
William.Smith@pnnl.gov

Deborah L. McGuinness
Tetherless World Constellation, Department of Computer Science, Rensselaer Polytechnic Institute
Troy, NY, USA
dlm@cs.rpi.edu

ABSTRACT

While streaming data have become increasingly more popular in business and research communities, semantic models and processing software for streaming data have not kept pace. Traditional semantic solutions have not addressed transient data streams. Semantic web languages (e.g., RDF, OWL) have typically addressed static data settings and linked data approaches have predominantly addressed static or growing data repositories. Streaming data settings have some fundamental differences; in particular, data are consumed on the fly and data may expire.

Stream reasoning, a combination of stream processing and semantic reasoning, has emerged with the vision of providing "smart" processing of streaming data. C-SPARQL is a prominent stream reasoning system that handles semantic (RDF) data streams. Many stream reasoning systems including C-SPARQL use a sliding window and use data arrival time to evict data. For data streams that include expiration times, a simple arrival time scheme is inadequate if the window size does not match the expiration period.

In this paper, we propose a cache-enabled, order-aware, ontology-based stream reasoning framework. This framework consumes RDF streams with expiration timestamps assigned by the streaming source. Our framework utilizes both arrival and expiration timestamps in its cache eviction policies. In addition, we introduce the notion of "semantic importance" which aims to address the relevance of data to the expected reasoning, thus enabling the eviction algorithms to be more context- and reasoning-aware when choosing what data to maintain for question answering. We evaluate this framework by implementing three different prototypes and utilizing five metrics. The trade-offs of deploying the proposed framework are also discussed.

Categories and Subject Descriptors

C.1.3 [**Other Architecture Styles**]: Data-flow architectures—*stream reasoning*; D.2.11 [**Software Architectures**]: Patterns

General Terms

Design, Experimentation, Management

Keywords

Data Cache, Order-awareness, Stream Reasoning, Semantic Web

1. INTRODUCTION

Streaming data are increasingly pervasive on the web, however many semantic applications may not be well aligned with the requirements of streaming data applications[17]. The Semantic Web[9] architecture has traditionally concentrated on storing and linking the web of data, rather than on managing rapidly changing data streams that become obsolete over time[16]. Although capable of processing data streams at large scale with a high velocity data rate, the primitive operations in the data-stream management systems[14] have not typically addressed the extraction of hidden knowledge via complex reasoning. In 2009, E. Della Valle et al.[15] introduced the research area of stream reasoning with the aim to bridge the gap between semantic reasoning and stream processing. Stream reasoning can provide many benefits in application areas that demand generation and analysis of data streams. Examples include smart cities[27][22], social networks[5] and financial market data feeds[26]. These data streams often have diverse conceptual models and physical formats and thus pose challenges for effective semantic processing and reasoning.

The best practices for linking static information on the web have inspired several solutions to face these challenges. Examples include extending RDF and Linked Data principles to model data streams, extending SPARQL to continuously process RDF streams, and implementing efficient stream reasoning systems.

1.1 RDF streams

In 2009, E. Della Valle et al.[15] envisioned two alternative RDF stream formats, namely the RDF molecules stream and RDF statements stream. The former is an infinite number of pairs $<\rho, \tau>$, where ρ is an RDF molecule[19], and τ is a timestamp denoting the arrival time of ρ; the latter is a special case of the former, where ρ only contains one statement.

D. F. Barbieri et al.[8] proposed an approach to publish data streams as Linked Data. In this work, an RDF stream is defined as an ordered sequence of pairs, each of which consists of an RDF triple and a monotonically non-decreasing timestamp τ.

An RDF stream[4] is identified by a unique IRI that is a streaming source locator, and is published in a named graph[13]. Each named graph is given an IRI that is designed following the guidelines of Cool URIs[25] and best practices on how to publish Linked Data on the Web[10].

RDF streams are generated and consumed in a certain order. This order can be a natural time-based order, or it may use another ranking criteria such as importance or precision. Thus, semantic RDF Stream Processing (RSP) systems should be able to manage data with rank-awareness and time sensitiveness.

1.2 Existing RDF Stream Processing Systems

C-SPARQL[4], among the initial RSP[1] languages, is a continuous extension of the standard SPARQL. It is tailored to semantically process data streams and facilitate reasoning. A C-SPARQL query is registered in a form of either a stream or a query, prior to the arrival of data streams. Its execution model is inherited from CQL[3], including operators of stream-to-relation, relation-to-relation and relation-to-stream. Its built-in translator will translate a C-SPARQL query into static and dynamic parts, and execute them separately. The C-SPARQL engine can be used as a linked data stream publisher[8].

Other works such as EP-SPARQL[2], TrOWL[28] and Stream SPARQL[12] are either extensions of SPARQL from different angles or are built from scratch to fulfill the purpose of continuously processing and reasoning on data streams. The IMaRS[6] algorithm has been proposed by the same authors of C-SPARQL, and is focused on inferred statement management (mainly statement deletions) in a time-based sliding window. This algorithm assigns an expiration timestamp for each RDF statement entering into the window, labels and updates all the related inferences with this expiration timestamp. The expiration timestamp is the time when the explicit data exit the window, and is calculated by adding the data arrival timestamp and the window size. A deletion is triggered when the original explicitly stated data exits the window and both explicit and inferred statements will be deleted. However, IMaRS is not adequate to process RDF streams with source-assigned expiration timestamps, because it cannot control when the data expire and lacks the ability to delete the data that expire before exiting the window. Hence, two problems will be caused. The First In First Out (FIFO) eviction strategy can evict unexpired data, data with a valid period longer than the window size, which still have the potential to contribute to the reasoning and query. The FIFO eviction strategy can also let expired data, data with a valid period shorter than the window size, generate invalid reasoning and query results in the window. Thus, not only the arrival order, but also the expiration order of RDF streams has a big effect on the RSP outputs.

1.3 Assumptions and Contributions

This paper proposes a cache-enabled, order-aware, ontology-based stream reasoning framework. This framework is able to process and reason across dynamic streaming data and provide correct results. It uses a background ontology that describes the domain knowledge where the streaming data is interpreted. It also leverages a data cache and a set of order-aware data management strategies. Our framework is built under the following assumptions.

- The background ontology is provided by domain experts and does not change during processing.
- The streaming sources encapsulate the streaming data in unique named graphs and assign expiration timestamps.
- An arrival timestamp is assigned to each streaming graph by the framework.

Under these assumptions, we list the following contributions.

- We leverage a data cache and order-aware algorithms[2] in a stream reasoning context to manage and process RDF streams.
- We define *semantic importance* as a ranking strategy and show its value to distinguish a cache from a window, facilitate order-awareness and data management in the cache.
- We implement three prototypes[3] based on off-the-shelf triplestores, and evaluate them under different cache configurations[4] from the following aspects:
  - runtime of query, reasoning, reasoning explanation and data eviction
  - the statistics of precision, recall and F-measure
- We discuss the trade-offs to deploy our framework in different scenarios where small, medium and large RDF streams are processed.

[1]https://www.w3.org/community/rsp/

[2]We introduce them in details in Section 3.

[3]https://github.com/raymondino/CacheStreamReasoning

[4]We cover cache configurations in Section 3.

2. APPROACH

We use the cache as our framework's central component to manage the RDF streams. Similar to what a window does in other RSP settings, the cache works as a stream-to-relation (S2R) operator[7], isolating a portion of the unbounded RDF streams. However, crucial differences between them lie in both their data consumption rules and data eviction strategies. The window consumes RDF streams by sliding along them. This restricts data eviction to only be FIFO, namely the window is only able to move forward by deleting some old data. The cache keeps static and is fed by RDF streams. It has flexibility to utilize semantics (the background ontology and any processing results) to inform its eviction and potentially consumption. All cached RDF streams can be ranked using different types of criteria. Temporal orders (arrival and/or expiration) are most common although other considerations including precision, popularity, trust, certainty, and provenance have been proposed [17]. We can leverage all of these rankings in our data eviction policies and in addition, we add reasoning perspectives to the criteria for ranking.

FIFO can be realized by ranking arrival timestamps. First Expired First Out (FEFO) can be realized by ranking expiration timestamps to guarantee valid results. Semantic ranks can also be provided from the reasoning and/or query participation status of the data.

In order to present these semantic ranks, we would like to introduce the notion of semantic importance. For each named graph in the cache, its semantic importance (SI) is defined as an indicator that measures its contribution to the reasoning and/or query results. The cache ranks all named graphs according to their SI. This notion is intentionally abstract so as not to limit its application in different data eviction strategies where a named graph's contributions are specified. For every named graph in FIFO, its SI is based on its arrival timestamp. For every named graph in FEFO, its SI is based on its expiration timestamp. The cache also uses Least-Frequently-Used and Least-Recently-Used algorithms to semantically rank the data. In Least-Frequently-Used, SI is embodied as a total reasoning-participation frequency counter of a named graph. In Least-Recently-Used, SI is embodied as the most recent reasoning-participation timestamp of a named graph. SI can also be composite under the scenario where multiple data eviction strategies are applied.

We have discussed above that there are distinctions between a window and a cache, and SI is among the keys to distinguish them. SI is mentioned in several published papers but never as a core topic. In[11], the authors use SI as an attribute of interface objects to improve the design of traditional GUIs in the Computer Human Interaction (CHI) domain. In[23], the authors use SI to filter Snooker balls in order to 3D reconstruct the game for analysis. In[31], the authors use SI to differentiate the semantic components in semantic-linked network. Nonetheless, none of these works formally define SI. This, together with the value that the SI provides in the stream reasoning context, contributes to our motivation for defining it for the stream reasoning context.

Our framework utilizes background knowledge encoded in OWL[24] when processing RDF streams. The ontology is loaded into the cache prior the data arrival, and provided by the domain experts. It describes the specific domain knowledge necessary to interpret the RDF streams. However, its expressiveness should be considered together with the query and deployed reasoner ability. If the ontology is in a description logic (DL) profile and the query requires DL reasoning, but the reasoner only provides RDFS reasoning, then the goal will not be achieved. Generally speaking, the purpose of the ontology is to provide enough domain background information and enable reasoning, which will help the query answer questions that cannot be answered directly from the explicit data.

2.1 Framework Architecture

Our framework consists of four sequential components: data consumption, querying and reasoning, reasoning explanation and data eviction. These four components are executed in order. The execution flow forms a loop to process data streams and to produce query results in a continuous fashion, as it is shown in Figure 1.

2.1.1 Data Consumption Component

In this work, we extend the RDF stream format by adding another timestamp and a unique graph ID. An RDF stream is represented by $<\rho, \tau_a, \tau_e, G>$, where ρ denotes the RDF molecule/statement, τ_a denotes its arrival timestamp, τ_e denotes its expiration timestamp and G denotes its unique graph ID. The RDF streams are first consumed by the Data Consumption Component (DCC)at Stage 1. DCC works like a storm spout. When the cache is not full, DCC sends RDF streams to it. When the cache is full, DCC stops sending. The incoming RDF streams will wait till the next cache opening. The data structure in the cache is a minimum heap, which enables the cache to be order-aware. The key feature of the minimum heap is that it always keeps the smallest element at the top. Evicting the top element takes O(1) time, then the minimum heap takes O(logn) time to update its top with the smallest element among the preserved ones. If we incorporate this feature with SI, the least semantically important data will be at the top and can be evicted easily. The bigger the SI is, the more important the data is, the less likely it is evicted.

The arriving data will be immediately ranked by the cache and ready for the next step.

2.1.2 Querying and Reasoning Component

When the cache is full, the processing moves to Stage 2 - Querying and Reasoning Component. The query in the stream reasoning scenario should be continuous in order to provide proactive answers. This requires the query to be pre-registered in the system before the data arrival. We use standard SPARQL query that will be executed continuously [5] in the framework , which is shown in Figure 1.

In our framework, reasoning happens during the query time. This provides an advantage that only the necessary entailments for the answer will be computed. However, we would like to point out that reasoning does not have to happen at the same time as the query. One example is materialization that is performed iteratively: the materialized snapshot of the database is always updated as long as the new data arrive.

[5]This continuous standard SPARQL is another key difference among C-SPARQL, EP-SPARQL and other extended SPARQL as the latters require different execution models and syntaxes so that the learning curve might be steep for users.

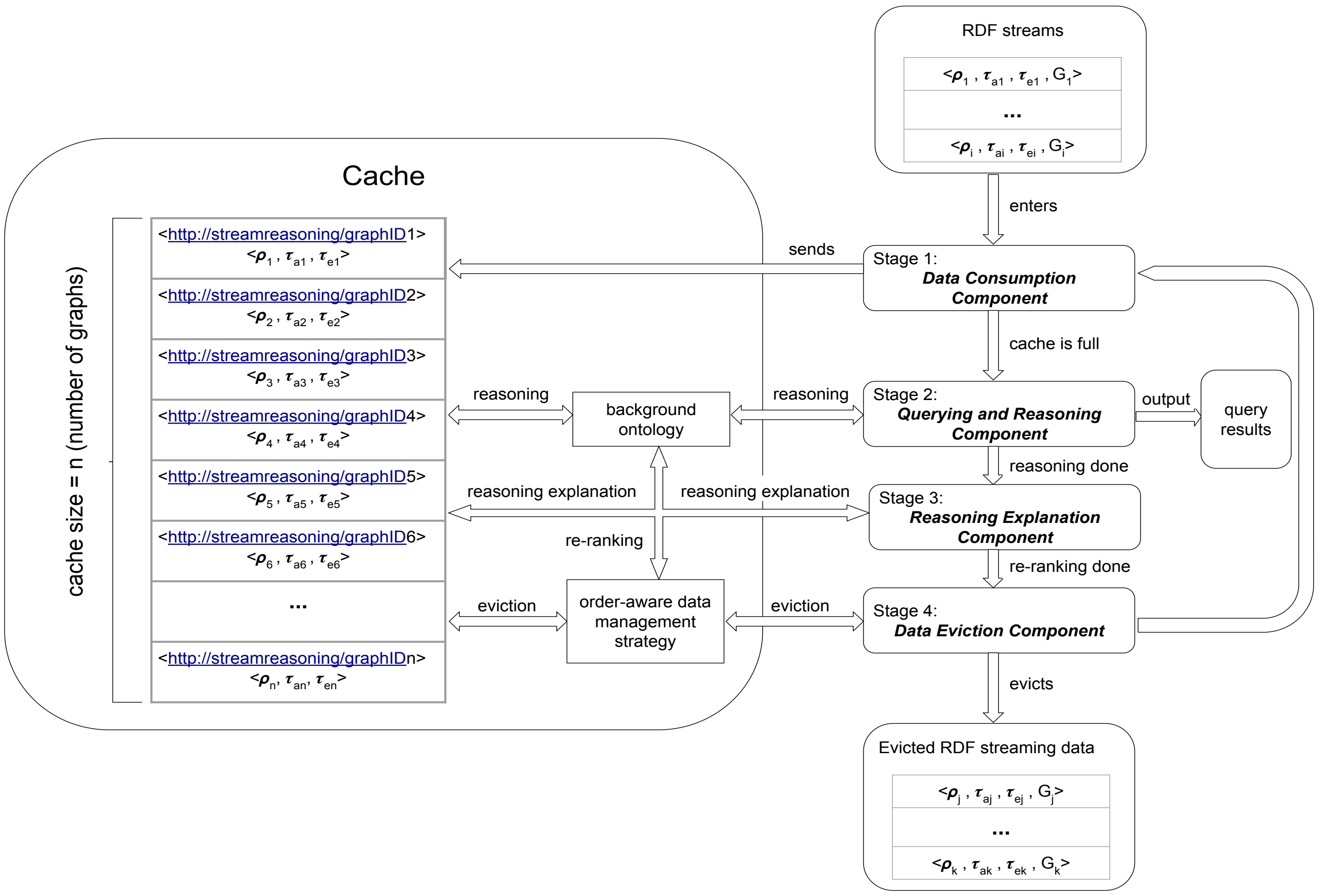
RDF streams
<ρ1 , τa1 , τe1 , G1>
...
<ρi , τai , τei , Gi>
enters
Cache
cache size = n (number of graphs)
<http://streamreasoning/graphID1> <ρ1 , τa1 , τe1>
<http://streamreasoning/graphID2> <ρ2 , τa2 , τe2>
<http://streamreasoning/graphID3> <ρ3 , τa3 , τe3>
<http://streamreasoning/graphID4> <ρ4 , τa4 , τe4>
<http://streamreasoning/graphID5> <ρ5 , τa5 , τe5>
<http://streamreasoning/graphID6> <ρ6 , τa6 , τe6>
...
<http://streamreasoning/graphIDn> <ρn, τan, τen>
sends
Stage 1: Data Consumption Component
cache is full
reasoning
background ontology
reasoning
Stage 2: Querying and Reasoning Component
output
query results
reasoning done
reasoning explanation
reasoning explanation
Stage 3: Reasoning Explanation Component
re-ranking
re-ranking done
eviction
order-aware data management strategy
eviction
Stage 4: Data Eviction Component
evicts
Evicted RDF streaming data
<ρj , τaj , τej , Gj>
...
<ρk , τak , τek , Gk>

Figure 1: Cache Enabled Stream Reasoning Framework Architecture.

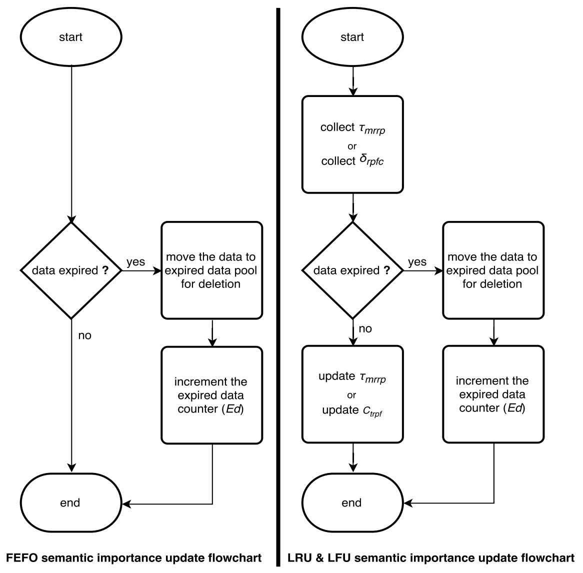
start
collect τ_{mrrp} or collect δ_{rpfc}
data expired ?
yes
no
move the data to expired data pool for deletion
increment the expired data counter (*Ed*)
update τ_{mrrp} or update C_{trpf}
end
FEFO semantic importance update flowchart
LRU & LFU semantic importance update flowchart

Figure 2: Semantic Importance Update Flowcharts

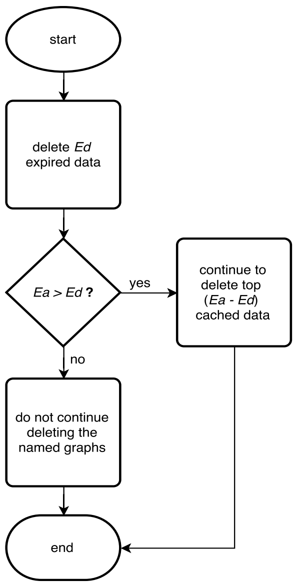
start
delete *Ed* expired data
Ea > Ed ?
yes
no
continue to delete top (*Ea - Ed*) cached data
do not continue deleting the named graphs
end

Figure 3: Data Eviction Flowchart

2.1.3 *Reasoning Explanation Component*

After the query results are delivered, the cache needs to know the reasoning-participation status of every named graph. In Stage 3, the Reasoning Explanation Component works in a way to trace back to the provenance of the inference; that is to find which cached named graphs participated in the reasoning process. The reasoning is explained by proof trees. A resulting inferred statement can be entailed by both explicit triples and intermediate inferences. Only explicit triples' graph IDs and reasoning-participation will be recorded. This provides the foundation to collect the statistics of the reasoning-participation for each of the named graphs. The statistics are different under different ranking algorithms, and we will cover the details in the Data Eviction Component.

2.1.4 *Data Eviction Component*

This component evicts the named graphs with small SI from the cache. We have applied three ranking algorithms shown in Table 1. One and only one ranking algorithm is applied in the cache at one time to determine which named graph needs to be evicted. In each framework execution cycle, every named graph's SI is updated by the statistics collected, as Figure 2 shows.

FEFO collects one statistic – the expired data count, E_d. E_d increments by one if a named graph is expired. The SI under FEFO is the expiration timestamp, which will not change. LRU collects two statistics – E_d and the most recent reasoning-participation timestamps, τ_{mrrp}, for every named graph. The SI is updated by incrementing E_d by one if a named graph is expired, and replacing every valid named graph's old τ_{mrrp} with the corresponding new τ_{mrrp}. LFU collects two statistics – E_d and every named graph's reasoning-participation frequency counts, δ_{rpfc}, in the latest cycle. The SI is updated in the following way. E_d increments by one if a named graph is expired. Every valid named graph's total reasoning-participation frequency counter, C_{trpf}, adds the corresponding δ_{rpfc}.

The cache re-ranks the data immediately after the SI is updated. We introduce a parameter called eviction amount, E_a, to guarantee a minimum percentage of the cache is available for the insertion of new data.[6] When composite SI, like expiration timestamp and reasoning-participation frequency, is applied, it is possible that data with long expiration timestamps do not participate in the reasoning at all. While valid, this data are less semantically important, and the framework should make this data available for eviction, till the actual amount of data evicted is equal to E_a.

Figure 3 shows the data eviction procedure in this component. The cache will first delete all data in the expired data pool (that is E_d data). Then a comparison between E_a and E_d will be made. If E_a is greater, the cache will continue to delete top $(E_a - E_d)$ data. Otherwise, no data will be evicted.

2.2 Framework Implementation

We have implemented our framework using two off-the-shelf triplestores, AllegroGraph V5.0.2[7] and Stardog 4.0 RC3[8]. Both provide Java APIs and step-by-step tutorials. AllegroGraph is a modern and efficient graph database and a commercial product of Franz Inc[9]. It supports up to OWL2 RL reasoning and full SPARQL 1.1. Stardog is a graph database by Complexible Inc[10]. It supports up to OWL2 DL & rule-based reasoning and SPARQL 1.1. We use the free versions of both products.

Stardog has an important feature - the ability to support reasoning explanation. An entailment is explained by tracking back to the original asserted statements, which form a proof tree that includes all the statements involved during the reasoning. Stardog also supports merging proof trees because the same inferred statements can be generated utiliz-

[6]We cover the details of E_a in Section 3.
[7]http://franz.com/agraph/allegrograph/
[8]http://stardog.com/
[9]http://franz.com/
[10]http://complexible.com/

Table 1: Applied Ranking Algorithms

Algorithm	Abbreviation	Type	Semantic Importance
First Expired First Out	FEFO	temporal, single	expiration timestamp
FEFO & Least Recently Used	LRU	temporal, composite	expiration timestamp & most recent reasoning-participation timestamp
FEFO & Least Frequently Used	LFU	counter-based, composite	expiration timestamp & total reasoning-participation count

ing different reasoning paths. This allows us to analyze and rank the reasoning-participation of each data in the cache. Unfortunately, AllegroGraph does not provide similar functionality thus we can only manage data under the FEFO strategy with it.

The cache is implemented in the triplestore. It has a fixed size, C_s, limiting the contained maximum data amount. Eviction amount, E_a, as mentioned in Section 2, denotes the least amount of data to be deleted in each framework execution cycle. The *SPARQL drop* argument is leveraged to fully remove named graphs from the cache in the data eviction component. We avoid using the *SPARQL delete* because it only removes the statements in the graphs, not the graph ids. This will pollute the cache.

Stardog supports both memory & disk-based databases, while AllegroGraph only supports disk-based databases. We have implemented three prototypes, focusing on Stardog memory & disk-based and AllegroGraph disk-based cache stream reasoning system. As we have already mentioned, the reasoning abilities of these two triplestores are different. In order to perform a fair comparison we use a query that only requires RDFS reasoning[11].

3. EVALUATION

The stream reasoning community has not yet come to consensus on the best method to evaluate stream reasoning applications. In 2012 SRbench[30] was proposed as a general benchmark system designed to test streaming RDF/SPARQl engines. In the same year, LSBench[21] was proposed to focus on assessing different Linked Stream Data (LSD) applications' capabilities. The following year CSRBench[18] was proposed with a special emphasis on the effects of operation semantics for stream reasoning applications. More recently CityBench[1] was proposed to target smart city applications. We chose the LUBM benchmark[20] dataset because it is easy to use and satisfies our needs. LUBM provides a well-constructed ontology describing the relations among universities, professors and students etc. It also features a data generator which accepts customized parameters to generate arbitrary ABox data.

Our work produced 6,031,109 ABox triples. A data source generates streaming data by disk-reading this generated data line-by-line from a static n-triples file. We configured the streaming throughput as follows:

- The streaming source packs either 1, 10 or 100 triples per named graph, T_{pg}.
- Each graph is assigned a unique graph id and expiration timestamp by the streaming source[12].

[11]The team used Stardog to test OWL-DL reasoning on streaming data[29], but these results are not included within the scope of this paper.

[12]http://streamreasoning.org/slides/2015/10/sr4ld2015-02-

```
PREFIX rdf : < http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX lubm: < http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
SELECT DISTINCT ?s
WHERE {
    ?s rdf:type lubm:Professor .
}
```

Figure 4: The Pre-registered SPARQL Query that Requires RDFS Reasoning

- The framework assigns an arrival timestamp to each arriving named graph in a monotonically non-decreasing order.

The streaming data is then streamed to our three prototypes, where the cache is configured as follows:

- C_s is either 10, 100 or 1000 graphs.
- E_a is either 25%, 50%, 75% and 100% of C_s.
- The pre-registered query, as shown in Figure 4, requires RDFS reasoning.
- The background ontology is pre-loaded into the cache

Our evaluation platform specifications include 14.04 64bits Ubuntu LTS operating system, Intel(R) Xeon(R) CPU E5-2620 v2 @2.10GHz, 2040MB memory, and 16GB HDD.

Together with the generated ABox data and provided LUBM ontology we conducted 252 experiments and obtained a ground truth of 27,192 results as the basis to test the framework's correctness on precision, recall and F-measure. We also recorded each prototype's memory consumption, runtime of querying, reasoning explanation and data eviction under different configuration combinations of streaming data and cache. We did not record reasoning explanation time for all the FEFO strategy and other strategies with the $E_a = 100\%$ of C_s, because the FEFO does not need reasoning explanation, and it does not make sense to explain reasoning as the whole cache will be dumped.

Using these results we ask the following questions:

1. What are the effects of different caches from different producers (AllegroGraph v.s. Stardog), types (disk-based v.s. memory-based) and configurations (different combinations of C_s and E_a)?

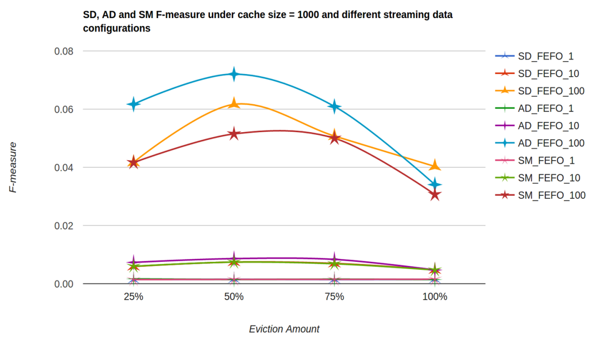

Figure 5: F-measure performance by different branded, typed and configured cache with FEFO strategy

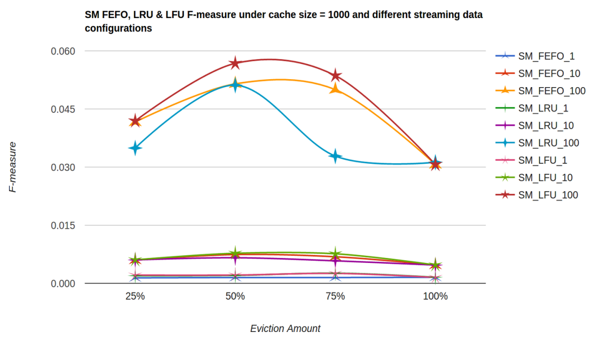

Figure 6: F-measure performance by different strategies under different configuration combinations

2. How do various strategies perform under different combined configurations of the streaming data and cache (currently we are using F-measure to evaluate the performance)?

3. What are the trade-offs to consider when deploying our framework in different scenarios?

We show two figures of the total forty-six visualizations[13] generated from the results to answer the first two questions. The third one will be thoroughly discussed in the next section.

For the sake of convenience, we use the following abbreviations in the rest of this paper: prototypes are abbreviated such that SM denotes Stardog memory-based cache, SD denotes Stardog disk-based cache and AD denotes AlleGroGraph disk-based cache. Each test case is labeled as $<$ prototype abbreviation $>$_$<$ data management strategy $>$_$<$ streaming data configuration $>$. For example, SD_FEFO_1 denotes a Stardog disk-based cache that performs FEFO strategy to process RDF streams with 1 triple per graph.

Figure 5 shows the F-measure performances brought by caches of different producers, types and configurations. There are several facts that can be easily captured:

- The F-measure increases as the streaming throughput increases.
- The F-measure at 50% is always best, and 100% is always worst for all visualized cases.

[13]Please refer to out github repository for all visualizations.

- AD_FEFO performs similarly as others do when streaming throughput is 1 triple per graph, but outperforms as the streaming configuration increases.
- SD_FEFO and SM_FEFO compete with each other in each test run.

This can partially answer the first question, with the points that different brands do affect the F-measure performance, but different types do not have a significant influence. The greater the streaming throughput, the more influences are made on the F-measure. However, in order to give a thorough answer we need to look at other metrics before assessing the overall performances of these caches. For example, AD_FEFO gives the best F-measure, but does it take more time to execute the query and data eviction?

Figure 6 shows the F-measure performance by different strategies for the SM cache with different streaming data configurations. The observed facts are:

- The F-measure score increases as the streaming throughput increases.
- 50% eviction amount has the biggest F-measure score, 100% has the smallest score for all the cases.
- For the same streaming data configuration, LFU always performs best, followed by LRU and then FEFO.

These observations can answer the second question from the perspective of big cache size. Nevertheless, does cache size affect these strategies' performances? Does it take longer time to explain all of the inferences? If yes, is it worthwhile to sacrifice system responsiveness for a better F-measure?

Additional observations are as follows:

- F-measure score: Our raw experimental results have shown that F-measure increases as the cache size increases. The F-measure score is also affected by different triplestores. We believe this is because Alllegro-Graph's and Stardog's inner processing engines and mechanisms are different [14].
- Memory consumption: when $C_s = 10$, cases of $T_{pg} = 1$ require 3 times on average the memory of bigger streaming configured cases. When $C_s = 100$, the overall memory consumption decreases as the eviction amount increases. When $C_s = 1000$, the overall memory consumption increases as the eviction amount increases. However, within this evaluation, no significant differences are observed between memory-based cache and disk-based cache. Small cache and streaming throughput cases usually requires more memory. Average FEFO strategy memory consumed is 41.93MB, LRU is 40.4MB and LFU is 39.59MB.
- Query time: AD requires the most query time for all cases. SD requires less time than AD but more than SM. The query time increases as cache size and streaming throughput increases. One potential reason for this is because a disk-based cache needs some IO time when executing a query, which takes more time than a memory-based cache. Average query time for all FEFO cases is 18ms, LRU is 15ms, LFU is 13ms.

[14]Exploration and explanation of triplestores' inner implementations are out of the scope of this paper

- Explanation time: reasoning explanation time increases approximately linearly as cache and streaming throughput increases. In most cases, LFU takes longer time to explain. Average LFU explanation time for all LFU-cases is 90371ms, LRU is 86100ms.

- Eviction time: eviction time increases as eviction amount, cache size and streaming throughput increases. AD eviction time is very fast, 30ms on average. SD on average is 2879ms. SM on average is 4042ms.

4. DISCUSSION

According to the cache and streaming data configurations, there can be either 10, 100, 1,000, 10,000 or 100,000 triples in the cache during one processing loop. We identify a small case where 10 or 100 triples are processed, a medium case where 1,000 or 10,000 triples are processed, and a large case where 100,000 triples are processed. Together with Table 2 We present a thorough comparison among triplestores, cache types and data management strategies under these scenarios.

We summarize our experimental results in Table 2 to help answer the third question in the previous section, i.e., what are the trade-offs to consider when deploying our framework in different scenarios?

Under the small case, AllegroGraph's query and eviction time is several times that of Stardog. Disk and memory performs equally on F-measure, though disk requires more time to query, evict and explain. LFU performs best in F-measure, followed by LRU then FEFO. Though FEFO needs more time to query and evict, the explanation time required by LRU and LFU is significantly greater. The trade-offs in deploying our framework under the small scenario is dependent on the use case. If system responsiveness is the first class citizen, a FEFO strategy will be chosen since it does not require explanation and provides a fine F-measure. Stardog memory cache can be chosen since it provides faster execution time and better F-measure. If F-measure is most important, LFU is the right strategy. Stardog memory cache is the best as memory cache provides less explanation time. It is also noticed that $E_a = 25\%$ or 50% of C_s provides better F-measure.

For medium cases, Stardog's eviction time increases significantly. Though Stardog's query time is better than AllegroGraph's, the difference is very small when compared with the eviction time. Disk cache provides less eviction time; its other metrics are similar as memory caches. LFU is the best at F-measure scores, but is traded for longer explanation time. Actually FEFO provides decent F-measure without explanation time. Overall, AllegroGraph disk cache with FEFO is most suitable for this scenario. $E_a = 50\%$ or 75% of C_s provides best F-measure as well.

For large cases, AllegroGraph's eviction time, F-measure and memory performs better, though query is slower. Disk cache query time is 9 times greater than the memory dependent graph database, but provides better F-measure, explanation , eviction time and memory. FEFO performs best in F-measure but it spends more time on query, with smallest eviction time and memory consumption. Hence, AllegroGraph disk cache with FEFO is most suitable for this experiment's use case. It is also recommended to use $E_a = 50\%$ of C_s to provide the best F-measure.

5. CONCLUSIONS AND FUTURE WORK

We have discussed the distinctions between a window and a cache, highlighted some challenges with a simple sliding window, and have described the enhanced data eviction flexibility that a cache is able to provide. We have defined semantic importance as a ranking strategy and showed how it can be exploited in a stream-reasoning context. We implemented semantic importance with a range of settings and evaluated them, laying the foundation for expanding the range of data management strategies. We have also presented our cache-enabled stream reasoning framework. By leveraging a cache and a set of data management strategies, this framework is able to consume RDF streams on the fly while performing reasoning and answering standing queries. The cached RDF streams are ranked according to the semantic importance, and data are evicted when as less important or expired. We have implemented three prototypes of the proposed framework, and evaluated them by emulating an RDF stream generated by LUBM and time stamped with arrival and expiration times. We also discussed the trade-offs of deploying our framework in different scenarios.

In this work, our framework and prototypes are evaluated on a synthetic situation. Future work includes applying our framework on realistic data set benchmarks such as SRbench, as well as apply our work on some compelling use cases such as smart cities.[15]

Our current data eviction strategies are domain agnostic. We will also develop some domain literate strategies for our actual scenarios. (We have partially applied the framework in one streaming NMR setting with one simple domain-aware strategy yielding promising results). We noticed that the reasoning environment in this work is limited due to AllegroGraph's limited reasoning ability, and would like to explore more complex reasoning in OWL DL and/or rule-based reasoning as our future work as well.

6. ACKNOWLEDGMENT

The research described in this paper is part of the Analysis in Motion Initiative at Pacific Northwest National Laboratory. It was conducted under the Laboratory Directed Research and Development Program at PNNL, a multi-program national laboratory operated by Battelle for the U.S. Department of Energy.

We would like to thank Dr. Mark T. Greaves from PNNL for his valuable thoughts and discussions that helped shape the work and this paper. We also want to thank Dr. Emanuele Della Valle from the Politecnico di Milano for his inspiring discussions and encouragement to this work.

7. REFERENCES

[1] M. I. Ali, F. Gao, and A. Mileo. CityBench: A Configurable Benchmark to Evaluate RSP Engines Using Smart City Datasets. In *The Semantic Web-ISWC 2015*, pages 374–389. Springer, 2015.

[2] D. Anicic, P. Fodor, S. Rudolph, and N. 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.

[15] We plan to use Dublin City datasets from http://dublinked.ie/

Table 2: The Trade-off Table

Scenario	Triples in the Cache	Cache Configurations	SPARQL (ms)	F-measure	Explain (ms)	Eviction (ms)	Memory (MB)
Small	10	AllegroGraph	20.25	4.35E-05	-	69.25	22.00
		Stardog	5.13	3.93E-05	-	11.00	56.75
		Disk	8.69	5.43E-05	1970.25	26.94	57.00
		Memory	5.00	5.40E-05	1790.25	9.50	56.25
		FEFO	10.17	4.07E-05	0	30.42	45.17
		LRU	5.00	5.68E-05	1571.00	11.50	61.38
		LFU	4.63	7.19E-05	2189.50	11.50	74.75
	100	AllegroGraph	11.63	2.97E-04	-	73.38	58.38
		Stardog	5.00	3.05E-04	-	20.81	42.81
		Disk	6.75	3.70E-04	8058.81	34.94	41.88
		Memory	4.92	3.84E-04	8178.63	20.17	41.08
		FEFO	5.41	2.27E-04	0	28.75	36.00
		LRU	5.41	4.17E-04	7846.44	21.38	37.19
		LFU	5.00	4.44E-04	8391.00	21.25	36.19
Medium	1,000	AllegroGraph	12.17	1.53E-03	-	89.42	59.08
		Stardog	6.46	1.55E-03	-	164.96	36.21
		Disk	8.35	1.68E-03	25372.54	129.35	40.52
		Memory	6.56	1.74E-03	25265.42	169.14	36.97
		FEFO	8.36	1.54E-03	0	139.78	43.53
		LRU	6.96	1.82E-03	24355.21	151.33	37.54
		LFU	7.04	1.85E-03	26282.75	151.42	33.21
	10,000	AllegroGraph	15.63	7.25E-03	-	90.00	33.38
		Stardog	9.56	6.49E-03	-	1575.38	37.93
		Disk	11.34	6.77E-03	122612.38	1020.72	39.41
		Memory	9.22	6.77E-03	144213.08	1150.50	36.28
		FEFO	11.58	6.75E-03	0	1080.25	36.42
		LRU	9.50	6.18E-03	113897.38	1556.94	40.81
		LFU	9.81	6.89E-03	113782.88	1004.19	32.13
Large	100,000	AllegroGraph	127.75	5.72E-02	-	187.25	23.50
		Stardog	76.13	4.60E-02	-	27949.00	36.13
		Disk	134.31	4.64E-02	261564.75	17122.81	31.13
		Memory	15.17	4.22E-02	271454.38	32248.08	36.67
		FEFO	93.33	4.97E-02	0	18695.08	31.92
		LRU	84.88	3.57E-02	263054.63	27104.50	33.63
		LFU	66.50	4.58E-02	269964.50	27470.63	35.75

Each value in the table is calculated from raw experimental results. Please refer to our github repository for these raw results. In the 10 triples/cache scenario, for example, disk-based cache's F-measure score is averaged from AD_FEFO, SD_FEFO, SD_LRU & SD_LFU test cases, while memory-based cache's score is averaged from SM_FEFO, SM_LRU & SM_LFU test cases.

We would like to highlight an empirical comparison of the two cache types, the disk-based cache (DC) and the memory-based cache (MC).

- SPARQL query time: for every scenario, DC is slower than MC. As the scenario increases, DC grows much faster than MC. The ratio between 100,000 triples/cache and 10 triples/cache for DC is 15.45, whilst MC is 3.34. Though DC is less restricted by storage space, its query time slows down the system response time, and this effect will become worse as the scenario size grows. However, this is expected, as accessing disk when executing query is always slower than accessing the memory.
- F-measure: for every scenario, DC's and MC's F-measure scores are very similar. This means both types are able to provide same level correctness. F-measure increases as scenario increases, this is because the more data in the cache, the more correct results can be calculated, which raises the F-measure.
- Reasoning Explanation Time: in most scenarios, MC's explanation time is slower than DC's. The difference increases significantly as the scenario size increases. Reasoning explanation is very time-consuming. Strategies (such as LFU) requiring reasoning-explanation provide better F-measure when the scenario is small and explanation time is quick, but provide similar F-measure when the scenario is large and explanation time is slow.
- Eviction Time: in small scenarios, DC is slower; in medium and large scenarios, MC is slower. The difference between DC and MC increases significantly as the scenario size increases. This indicates a bias towards DC for large RDF streams, and MC for small RDF streams.
- Memory Consumption: Both DC and MC consume similar memory for each scenario. This could be because Stardog triplestores are implemented very efficiently, but again, explaining this requires the knowledge of inner mechanisms of Stardog, which is out of the scope of this work.

[3] A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal - The International Journal on Very Large Data Bases*, 15(2):121–142, 2006.

[4] D. F. Barbieri, D. Braga, S. Ceri, E. Della Valle, and M. Grossniklaus. C-SPARQL: SPARQL for continuous querying. In *Proceedings of the 18th international conference on World Wide Web*, pages 1061–1062. ACM, 2009.

[5] D. F. Barbieri, D. Braga, S. Ceri, E. Della Valle, and M. Grossniklaus. Continuous queries and real-time analysis of social semantic data with c-sparql. In *Proceedings of Social Data on the Web Workshop at the 8th International Semantic Web Conference*, volume 10, 2009.

[6] D. F. Barbieri, D. Braga, S. Ceri, E. Della Valle, and M. Grossniklaus. *Incremental reasoning on streams and rich background knowledge*. Springer, 2010.

[7] D. F. Barbieri, D. Braga, S. Ceri, and M. Grossniklaus. An execution environment for C-SPARQL queries. In *Proceedings of the 13th International Conference on Extending Database Technology*, pages 441–452. ACM, 2010.

[8] D. F. Barbieri and E. Valle. A proposal for publishing data streams as linked data. In *Linked Data on the Web Workshop*, 2010.

[9] T. Berners-Lee, J. Hendler, O. Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[10] C. Bizer, R. Cyganiak, T. Heath, et al. How to publish linked data on the web. 2007.

[11] R. Blanch, Y. Guiard, and M. Beaudouin-Lafon. Semantic pointing: improving target acquisition with control-display ratio adaptation. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 519–526. ACM, 2004.

[12] A. Bolles, M. Grawunder, and J. Jacobi. *Streaming SPARQL-extending SPARQL to process data streams*. Springer, 2008.

[13] J. J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named graphs, provenance and trust. In *Proceedings of the 14th international conference on World Wide Web*, pages 613–622. ACM, 2005.

[14] G. Cugola and A. Margara. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys (CSUR)*, 44(3):15, 2012.

[15] E. Della Valle, S. Ceri, D. F. Barbieri, D. Braga, and A. Campi. *A first step towards stream reasoning*. Springer, 2009.

[16] E. Della Valle, S. Ceri, F. Van Harmelen, and D. Fensel. It's a streaming world! Reasoning upon rapidly changing information. *IEEE Intelligent Systems*, (6):83–89, 2009.

[17] E. Della Valle, S. Schlobach, M. Krötzsch, A. Bozzon, S. Ceri, and I. Horrocks. Order matters! Harnessing a world of orderings for reasoning over massive data. *Semantic Web*, 4(2):219–231, 2013.

[18] D. Dell'Aglio, J.-P. Calbimonte, M. Balduini, O. Corcho, and E. Della Valle. On correctness in rdf stream processor benchmarking. In *The Semantic Web–ISWC 2013*, pages 326–342. Springer, 2013.

[19] L. Ding, T. Finin, Y. Peng, P. P. Da Silva, and D. L. McGuinness. Tracking rdf graph provenance using rdf molecules. In *Proc. of the 4th International Semantic Web Conference (Poster)*, page 42, 2005.

[20] Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2):158–182, 2005.

[21] D. Le-Phuoc, M. Dao-Tran, M.-D. Pham, P. Boncz, T. Eiter, and M. Fink. Linked stream data processing engines: Facts and figures. In *The Semantic Web–ISWC 2012*, pages 300–312. Springer, 2012.

[22] F. Lécué, S. Kotoulas, and P. Mac Aonghusa. Capturing the pulse of cities: Opportunity and research challenges for robust stream data reasoning. In *Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[23] P. Legg, M. L. Parry, D. H. Chung, R. M. Jiang, A. Morris, I. W. Griffiths, D. Marshall, M. Chen, et al. Intelligent filtering by semantic importance for single-view 3d reconstruction from snooker video. In *Image Processing (ICIP), 2011 18th IEEE International Conference on*, pages 2385–2388. IEEE, 2011.

[24] D. L. McGuinness, F. Van Harmelen, et al. OWL web ontology language overview. *W3C recommendation*, 10(10):2004, 2004.

[25] L. Sauermann, R. Cyganiak, and M. Völkel. Cool URIs for the semantic web. 2011.

[26] M. Stonebraker, U. Çetintemel, and S. Zdonik. The 8 requirements of real-time stream processing. *ACM SIGMOD Record*, 34(4):42–47, 2005.

[27] S. Tallevi-Diotallevi, S. Kotoulas, L. Foschini, F. Lécué, and A. Corradi. Real-time urban monitoring in dublin using semantic and stream technologies. In *The Semantic Web–ISWC 2013*, pages 178–194. Springer, 2013.

[28] E. Thomas, J. Z. Pan, and Y. Ren. TrOWL: Tractable OWL 2 reasoning infrastructure. In *The Semantic Web: Research and Applications*, pages 431–435. Springer, 2010.

[29] S. WP and M. Borkum. Semantically enabling stream-reasoning architectural frameworks. Abstract submitted to The 24th ACM International Conference on Information and Knowledge Management, Melbourne, Australia., 2015. PNNL-SA-110228.

[30] Y. Zhang, P. M. Duc, O. Corcho, and J.-P. Calbimonte. Srbench: a streaming rdf/sparql benchmark. In *The Semantic Web–ISWC 2012*, pages 641–657. Springer, 2012.

[31] H. Zhuge and L. Zheng. Ranking Semantic-linked Network. In *WWW (Posters)*, 2003.