Eviction Strategies for Semantic Flow Processing^{*}

Minh Khoa Nguyen,¹ Thomas Scharrenbach,¹ Abraham Bernstein¹

University of Zurich, Switzerland {nguyen,scharrenbach,bernstein}@ifi.uzh.ch

Abstract. In order to cope with the ever-increasing data volume continuous processing of incoming data via Semantic Flow Processing systems have been proposed. These systems allow to answer queries on streams of RDF triples. To achieve this goal they match (triple) patterns against the incoming stream and generate/update variable bindings. Yet, given the continuous nature of the stream the number of bindings can explode and exceed memory; in particular when computing aggregates. To make the information processing practical Semantic Flow Processing systems, therefore, typically limit the considered data to a (moving) window. Whilst this technique is simple it may not be able to find patterns spread further than the window or may still cause memory overruns when data is highly bursty.

In this paper we propose to maintain bindings (and thus memory) not on recency (i.e., a window) but on the likelihood of contributing to a complete match. We propose to base the decision on the matching likelihood and not creation time (fifo) or at random. Furthermore we propose to drop variable bindings instead of data as do load shedding approaches. Specifically, we systematically investigate deterministic and the matching-likelihood based probabilistic eviction strategy for dropping variable bindings in terms of recall. We find that a matching likelihood based eviction can outperform fifo and random eviction strategies on synthetic as well as real world data.

Keywords: Semantic Flow Processing, Linked Data stream processing, eviction strategies, load shedding, matching likelihood estimation on streams

1 Introduction

Information processing increasingly incorporates query matching on dynamically changing information (i.e. information flows) [5]. Based on these Information Flow Processing (IFP) systems, such as C-SPARQL [4], EP-SPARQL [1], or CQELS [10] emerged that are capable of processing flows of semantically annotated data. All of these Semantic Flow Processing (SFP) systems allow defining

^{*} The research leading to these results has received funding from the European Union Seventh Framework Programme FP7/2007-2011 under grant agreement no 296126.

queries in a language that extends SPARQL with algebra adjustments for matching queries on flows of time-annotated RDF triples.

Each SFP system works on limited memory resources. They constrain the number of valid partial variable bindings (i.e., bindings that are not yet complete solutions to a query) by applying a window on the stream. Such a window limits the scope of query matching to, for example, a period of time or to a certain number of triples. On the one hand this limits the scope of answers the system can find. If a match requires two data items that do not fall in the same scope the system will never be able to perform such a match. On the other hand, even the limited scope can not necessarily guarantee that the system will never exceed its memory resources. Consider the situation when a popular TV broadcast/show, such as a football match, starts and many users switch to it at the same time. In this case we may find that a sudden burst in the input data could cause the system to exceed its resources. The situation gets worse when we are not only interested in the number of viewers for a show but want to track the users – the literature refer to the latter case as non-shrinking semantics [3].

To our knowledge no current SFP approach has a solution for freeing memory in case these exceed the available resources. While proposals have been made for load-shedding—dropping or ignoring incoming data—in IFP systems, only one of them investigated load-shedding according to a statistical model of the matching history [6]. We believe that instead of ignoring incoming data one should one should drop variable bindings based on their likelihood that they lead to a result. We refer to this as eviction strategies in contrast to load shedding.

Eviction strategies are more flexible than load shedding. If we drop a data item we may drop information too early in the matching process. Eviction allows us to define constraints at which point of the matching process we want to delete items. While there exist deterministic approaches that allow for dropping variable bindings, such as *first-in-first-out (fifo)* we propose to use the matching likelihood as the dropping criterion. In this paper we, hence, investigate the performance of different eviction strategies. Note that an eviction strategy is not the same as a garbage collection. While the latter removes those variables bindings that will not contribute to a solution, the former removes variable bindings that have the potential but cannot be kept due to memory limitation. As a consequence applying an eviction strategy could lead to incomplete results for the sake of feasibility.

In this paper we propose to use a probabilistic based eviction strategy that estimates the likelihood of a binding to contribute to a result in contrast to using a fixed time window for load shedding. Note that this paper intends to show that a probability based model can outperform fifo and random load shedding but does not provide a model of how to estimate those probabilities on the go. We investigate the effect our approach has on the completeness of results compared with deterministic approaches. We systematically investigated the challenges arising from the limited memory resources of regular joins for SFP systems. If the partial bindings for the potential join partners exceed the memory we start dropping partial results. Our findings enable an SFP to trade off completeness versus resources. To ensure methodological correctness we followed the PCKS approach [11] for defining stress tests for SFP systems.

As the relevant properties of an SFP system we identify completeness as a Quality of Service constraint. The corresponding challenge we want to tackle is the ability to handle bursts in the data. We therefore define recall as the key performance indicator for the evaluations of the tests we performed in the scope of this paper. In order to test the system we check how the system behaves under restricted resources and under unrestricted resources.

In our experiments we compared our probabilistic methods based on the matching likelihood of partial bindings with the first-in-first-out (fifo) deterministic method as well as random load shedding. We show that our statistical approach can outperform fifo and random load shedding both on simulated data as well as on a real-world dataset of viewership behavior of IPTV users. The improvement ranges from 14% to 50% when using synthetic and real world data.

This remainder of this paper is structured as follows: next we formally specify the context of SFP and then introduce our load-shedding approach. We then evaluate our approach on synthetic and real world data in Section 3. After a discussion of the results and limitations we present related work in Section 5 and conclude in Section 6.

2 Method: Likelihood-based Eviction of Partial Matches

This section first lays the formal groundwork for defining what eviction strategies are. Then, it introduces our likelihood-based eviction strategy as well as the baseline first-in-first-out (fifo) and random strategies.

2.1 Semantic Flow Processing

SFP rely on an algebra that modifies the SPARQL algebra for query matching on flows of data. The result of an algebra operation is a set of variable bindings consisting of a finite number of pairs ?var/val, where ?var is a variable name and val is an RDF term. A query is a tree of algebra expressions where results (i.e., bindings) are propagated from the leaves to the root. An SFP system, hence, consumes time-stamped RDF triples $\langle s, p, o \rangle$ [time] that are then consumed (or evaluated) by the algebra operations defined in the query tree. Hereby the system updates the bindings for all affected levels of the query tree and bindings at the root level will be emitted as results.

In our discussions we rely upon the definitions of SPARQL and RDF as given in the SPARQL-query specification [8]. Whilst we limit our discussions to joins and aggregates our approach is by no means limited to these operations. A *semantic flow* \mathcal{F} is a finite set of time-stamped RDF triples. We denote algebra expressions by α . The evaluation of an algebra operation α and a finite set of variable bindings Ω on a semantic flow again produces a (possibly new/updated) set of variable bindings $eval(\alpha, \Omega, \mathcal{F}) = \Omega'$. In a query tree all algebra expressions consume variable (a semantic flow) bindings, perform the corresponding operations, and then emit variable bindings.¹

SPARQL query processing systems usually implement an iterator-based approach: since they know all relevant data beforehand, they can successively perform the algebra operations recursively and, hence, know a binding was completely processed by an algebra expression. A SFP system, in contrast, has to keep its variable bindings to guarantee completeness. Consuming data in a streaming fashion turns the execution model upside-down. Instead of recursively processing all available data for each algebra expressions we have to continuously evaluate newly arriving data for all algebra expressions. Consequently, the algebra expressions in an SFP have never finished evaluating all available data and have to continuously maintain a list of corresponding variable bindings. Consider the simple join of two graph patterns ?a ab ?b and ?b bc ?c, for example. Assuming that the SFP system matched the first pattern with the binding variable binding $\mu = \{?a/a_1, ?b/b_1\}$ it is not really possible to say if and when a match for the second graph pattern might happen.

Keeping all variable bindings, however, will eventually exceed the system's memory resources and make processing unfeasible. Therefore, SFP systems typically limit the number of data they process by windowing, which limits the data considered for computing partial matches to a given period of time or number of triples. This allows to mark bindings with an an expiration time-stamp. A garbage collection process can then remove those variable bindings that become out-of-scope.

Unfortunately, windowing has its distinct problems. For example, queries that require to match patterns that unfold over periods longer than a practical window for example cannot be matched. In those cases there are two options: First one can apply load shedding – an approach that drops data from the input stream, for example by only considering every x-th data item for processing. Load shedding is a well-established approach and we discuss its difference to eviction in Section 5 along with related work.

Second, we may delete variable bindings instead of input data. In this paper paper we investigate this alternative approach we refer to as *eviction*. It leaves the input stream untouched but intermediate variable bindings get deleted to make place for new ones. In the following we discuss this approach in detail.

2.2 Eviction: Dropping Variable Bindings

In order to limit memory usage of a SFP system we propose to use eviction, which deletes partial binding from the processing tree. When deleting partial matches (or shedding load in the input stream) we potentially sacrifice completeness. As a consequence, it is imperative to understand what consequence eviction has on completeness as a Quality of Service (QoS) criteria. In this study we, therefore, compare the performance of the following *eviction functions*:

¹ Note that the leaves of the tree typically consume \mathcal{F} and that the root may emit RDF data to a new flow \mathcal{F}' .

- random deletion,
- time-based deletion (e.g., delete the oldest bindings first), and
- data-driven deletion of bindings.

While the first is self-explanatory, the latter two require further explanation.

Time-Based Deletion Time-based deletion bases on the assumption that the probability that a binding contributes to a result decreases with the amount of time it stays in the bindings cache. Whilst we know of no empirical evidence that this assumption holds for typical SFP benchmarks it is used in systems such as SASE+ [7], where it is referred to as least-recently-used (LRU).²

LRU can be implemented as a priority queue, which can be implemented via a heap. Heap organization has a complexity of $\mathcal{O}(\log n)$, where n is the number of elements in the bindings cache.

Note that the actual performance of LRU can depend on the time model for the bindings. A time model is either based on system time or on data time or on both. Furthermore, an LRU eviction function must specify which time-stamp of a binding it uses. This time-stamp can be the creation time of the binding or the time it was last updated. In the scope of this work we use data time and the time-stamp of the last update of the binding.

Data-Driven Eviction Data-driven eviction—the core contribution of this paper—bases the eviction decision of a binding μ on the probability that μ contributes to a result. This requires an estimation of the matching probability for μ . To that end we introduce the notion of the transactional closure for a binding μ in Section 2.3, which permits counting the number of results it contributed to for a certain part of the stream data.

Probability estimation can be performed either online or offline. In the online case the contribution probability has to be estimated along with the actual query processing, which is a prerequisite in cases where the data cannot be stored for offline analysis. In this paper we assume that we can compute the probabilities offline and describe the method for estimating the probabilities in Section 2.4.

When applying data-driven eviction on each operator's binding locally it requires the same computational effort as LRU.³ Akin to the LRU case, datadriven eviction can be implemented using a priority queue. The only difference is the comparison attribute which is now the matching probability rather than the time-stamp. Global data-driven eviction strategies may require additional effort (cf Section 2.5).

² Note that in our case LRU gives the same result as fifo since we only forward fresh bindings to next operator instead of performing updates on the current binding.

³ Assuming that the probabilities, as in our case, are computed offline and do not influence the online processing effort.

2.3 The Transactional Closure

In order to evaluate the performance of any eviction or load shedding operator we have to know whether deleting a data item or a variable binding impacts the results. A query result emerges from applying a set of algebra operators. In order to determine whether deletion of data items or variable bindings has an impact on the results of a query we hence need to know *all* sequences in which the application of a set of data items to the algebra operators of a query leads to a result.

We now define the basics for computing all possible paths to all query results for a flow \mathcal{F} . Any result of a query is a variable binding μ_{root} the root of the tree of the query's algebra expressions $\mathcal{T}_{\alpha_{\nabla \mathcal{U}\sqcup}}$. As such any of these root variable bindings emerges from a sequence of application of algebra expressions from $\mathcal{T}_{\alpha_{\nabla \mathcal{U}\sqcup}}$. As a result, for each μ_{root} we have a finite set of sets of variable bindings. We refer to this set as the *transactional closure* $(\mu)^+_{\alpha,\mathcal{F}}$ of the binding μ . Indeed, we can define such a transactional closure recursively for all bindings for all algebra expressions in $\mathcal{T}_{\alpha_{\nabla \mathcal{U}\sqcup}}$.

Having the definition of the transactional closure at hand we now investigate what happens, if we delete some of the variable bindings from the transactional closure. If we delete all entries from $(\mu)^+_{\alpha,\mathcal{F}}$, then the variable binding μ will never be created. In this study we want to know what impact any variable binding $\mu' \in (\mu)^+_{\alpha,\mathcal{F}}$ has on the transactional closure of μ .

We assume that each algebra expression α maintains a *bindings cache*: the finite collection of bindings, which we denote with Ω_{α} . If we now delete a variable binding μ' from Ω_{α} we would like to know which potential impact this deletion has on the query results. We hence define the transactional closure with respect to α , Ω , and \mathcal{F} by $(\mu)^+_{\alpha,\Omega,\mathcal{F}}$. It inductively defines those variable bindings that emerge from μ by applying α to $\langle \mathcal{F}, \Omega \rangle$. Let $R_{\alpha,\Omega,\mathcal{F}}$ be the result set of α for Ω and \mathcal{F} , then we call $R^{\mu}_{\alpha,\Omega,\mathcal{F}} = R_{\alpha,\Omega,\mathcal{F}} \cap (\mu)^+_{\alpha,\Omega,\mathcal{F}}$ the result set of μ .

The bindings cache of an algebra expression contains different sets of variable bindings. A join with *n* join partners, for example, has one set $\Omega_{\alpha,n}$ for each of the *n* join partners. In this study we assume that each of these sets can hold a limited number of variable bindings, i.e., $|\Omega_{\alpha,n}| \leq \omega_{\alpha,n}$, where $\omega_{\alpha,n}$ is the memory available to store bindings. When the application of α on the flow \mathcal{F} causes some of these sets to exceed its limit $(\omega_{\alpha,n})$, then we apply an eviction strategy **E** that removes variable bindings from $\Omega_{\alpha,n}$ such that the above condition will be fulfilled again.

2.4 Estimating Matching Probabilities

We estimate the selectivity of a variable binding μ by the number of results that depend on μ . We define the probability that the result set of a variable binding μ is non-empty as the above count normalized by the total number of results with respect to the bindings cache:

$$Pr(R_{\alpha,\Omega,\mathcal{F}} \neq \emptyset) = \frac{\left|R_{\alpha,\Omega,\mathcal{F}}^{\mu}\right|}{\left|R_{\alpha,\Omega,\mathcal{F}}\right|}$$

Note that the choice of α designates for which biding cache $\omega_{\alpha,n}$ this probability is computed.

The above formula, hence, computes the probability that there exists a binding μ' that bases on μ . If α is a query, then the above formula defines the probability that μ contributes to a query's root result.

In some cases it is possible that a binding μ' originates from two different upstream bindings. Consider, for example, the following operation joining two identical basic graph patterns $BGP(\mathbf{?x P Y})$. Here, any binding $\mu = \{\mathbf{?x/X}\}$ could originate from the first or the second triple pattern matching:

$$JOIN(BGP(\texttt{?x P Y}), BGP(\texttt{?x P Y}))$$

For the computation of the probabilities we count both original variable bindings.

2.5 Eviction Strategies: Local vs Global Approaches

A local eviction strategy operates locally on the binding cache of each operator. A global eviction strategy, in contrast, operates on all binding caches together, attempting to optimize the overall global performance. In other words, for a local eviction strategy all $\omega_{\alpha,n}$ are fixed whereas a global eviction strategy has the additional constraints that $\sum_{\alpha} \sum_{n} \omega_{\alpha,n} \leq \omega$ for all $\omega_{\alpha,n}$. A global eviction strategy hence requires to also solve (or approximate) a constraint optimization problem. Since it is our goal to show that data-driven eviction can lead to superior results over the other strategies we only have to show that one of the two approaches offers advantages over the other, non data-driven ones. We, hence, restrict ourselves to the simpler, local eviction strategy shown in Algorithm 1 and leave the global eviction strategy for future work.

Algorithm 1 Local eviction strategy for an algebra expression $\alpha = \alpha_1, \ldots, \alpha_I$, a flow \mathcal{F} , a set of sets of variable bindings Ω_{α_i, n_i} , limits on these ω_{α_i, n_i}

Apply \mathcal{F} to $\langle \alpha, \Omega \rangle$ for all $\Omega_{\alpha_i,n}$ do if $|\Omega_{\alpha_i,n}| > \omega_{\alpha_i,n}$ then $eviction(\Omega_{\alpha_i,n})$ end if end for

3 Evaluation

The main working hypothesis of our paper is that eviction based on the matching likelihood can outperform the fifo and random dropping approach of variable bindings. We therefore have to estimate the matching likelihood. This in turns required implementing the transactional closure in order to be able to compute the counts on which the matching likelihood is defined. In order to calculate the matching likelihood, we implemented an "oracle". This oracle allows to retrieve the cardinality of complete matches in the future. The oracle assumes that required bindings are not been evicted by other join operators. With this oracle we can compute the probabilities as proposed in Section 2.4. We will test probabilistic eviction with probabilities learned from past data in future studies. Please find the discussion of this limitation in Section 4.

In this section we evaluate our approach governed by the above hypothesis. We therefore compared the performance of our likelihood eviction with the *fifo* and *random* strategies with respect to recall as the key performance indicator (KPI).

3.1 Experimental Setup

We performed our experiments on a synthetic and a real world data set. We used the same query that performs the following join:

We considered joins without sequential or temporal constraints, because of memory constraints. Our goal was to compute the recall for all potential bindings. Computing the transactional closure for joins requires an exponential number of traces which matching originated from which variable bindings. We discuss this limitation in Section 4.

We performed the experiments considering local eviction. We tested the three different eviction strategies with bindings caches which we increased exponentially. We chose sizes as multiples of 10. The reason for this is that calculating matching statistics is currently a time consuming process and therefore we choose an exponentially increasing size of cache in the evaluation.

For the comparison we assumed the same time for determining whether an element was subject to eviction or not. This holds true for *fifo* and statistical eviction, as they both keep a cache where the elements to be evicted have to be sorted out. We can achieve this by applying Heap-Sort. We assume that the number of elements to be dropped by eviction to be constant in relation to the

size of the bindings cache. This way each eviction step is bounded by $\mathcal{O}(\log(n))$ where n is the size of the bindings cache.

All experiments were conducted using the UZH Katts Semantic Flow Processing engine.⁴ While we simulated an uniform distribution of results for the synthetic dataset we used anonymous IPTV viewership logs and joined them with Electronic Program Guide data. The data was gathered in the scope of the ViSTA-TV project.⁵ For our experiments we used a data sample from Aug 1st, 2012. The sample comprised about 300'000 events in total for the user log and the EPG data. We thereby only considered such properties that also occur in the query. For the above mentioned exponential explosion for tracing the matching we limited ourselves to a subset comprising 10'000 events.

To retrieve the exact matching probabilities for the bindings we stored those statistics in a sqlite3 in memory database. However, updating the matching probability for each binding every time unit (every time unit can change the likelihood of a match for a binding) is a very time expensive process. Therefore we updated the probability for each binding only once, namely when we added it to the bindings cache.

The following configuration was used to run all our experiments: 2 Quad-core Intel(R) Xeon(R) CPU X5570@ 2.93GHz, 72GB of RAM, 4 TB of disk space, running Fedora Core 12 kernel 2.6.32.14 64bit.

3.2 Results

As Figures 1-2 show, our likelihood based eviction approach always outperforms *fifo* and *random* eviction. The difference for the synthetic data set decreases with increasing size of the bindings cache from 50% to 10% difference. In the real world data set the differences in recall increases from 14% to 29% for the 10'000 events data set and for the 1'000 events data set we can observe a decrease from 32% to 14%.

We see that for the extreme size of the binding cache, we always obtain 100% recall for all approaches. We evaluate the overall performance of our approach by the recall measured for the bindings cache size that can hold up to 10% of the bindings of the cache for which we get 100% recall. While the matching likelihood based eviction performs very good on the synthetic data set (up to 91% recall with bindings cache of 10% relative of 100% performance limit), it performs not as good for the real-world data set (up to 60% recall with bindings cache of 10% performance limit).

4 Discussion

Experimental Results The results of our study indicates that likelihood based eviction can outperform *fifo* and *random*. This was the case for all experiments we

⁴ KATTS is a recursive acronym for Katts is A Triple Torrent Sieve and is accessible via GitHub at https://github.com/lorenzfischer/katts

⁵ http://vista-tv.eu



Fig. 1: Recall Generated Dataset with 145'000 events.

conducted. The relative difference between is larger for the synthetic data than for the real world data set. We think that our approach was able to better predict the matching likelihood for synthetic dataset as it was generated following some uniform distributions. Nevertheless an eviction strategy based on the matching likelihood seems to be a promising approach.

We also found that there is a relation between the size of the bindings caches and the recall. The larger the bindings cache the higher the recall. While this result seems not very surprising it yet indicates that future research will have to be performed finding out the exact size of the bindings cache such that the recall becomes 100%. Such research would also investigate at which cost an improvement of the recall using eviction strategies comes.

Shortcomings The most important shortcomings of our study are the limited number of data items we could process and the restriction to a query consisting of a regular join. The limitation to the small number of data items is caused by the exponential number of possible traces from variable bindings to results. Research on more complicated queries and larger data sets has hence to be performed online. This, in turn makes an exact calculation of the matching statistics infeasible. Therefore one has to learn the matching statistic which we tackle in our future work.

In this study we hence concentrated on investigating how well likelihood based eviction works in *—principle*, compared to *fifo* and *random* eviction. We are currently investigating how to best estimate the matching likelihood on the go, i.e., along with the processing of incoming data. Future experiments will



Fig. 2: Recall ViSTA-TV Dataset with 1'000 events.

hence systematically investigate different types of algebra expressions. These include aggregates but also joins with sequential and temporal constraints, as these are more typical for SFP systems. We believe that likelihood based eviction can help most for such cases where a lot of variable bindings are produced, e.g., in the case of aggregates, in particular for non-shrinking semantics.

Our simulation did not investigate response time or throughput but recall. While we showed that matching likelihood based eviction outperforms deterministic approaches like *fifo* and *random* eviction in terms of recall, we will investigate at which costs this comes for the former two KPIs. As in the previous case we will have to then estimate the statistics online rather than offline. Following the PCKS paradigm [11] are working on implementing tests for determining how well our approach works for bursts in the data.

In this study we considered a purely local eviction strategy. It would be interesting to see the effect of global constraints. We are currently investigating how to add these to statistical eviction. This requires a well-defined optimization strategy carefully balancing the relation between local and global constraints.

5 Related Work

Research related to this paper roughly covers three areas: Semantic Flow Processing, selectivity estimates on flow-data, and load shedding for IFP/SFP systems. In the sequel we discuss how our work relates to each of these.



Fig. 3: Recall ViSTA-TV Dataset with 10'000 events.

5.1 Semantic Flow Processing

C-SPARQL [4] performs query matching on subsets of the information flow defined by windows. The decidability of SPARQL query processing on such finite sets of RDF triples causes the number of variable bindings produces to be finite. Their number may still become prohibitively large, for example, when using non-shrinking semantics for aggregates [3]. Since the execution is performed on traditional SPARQL engines, we may apply our approach, for example as a filter for iterator-based implementations such as Jena.⁶

EP-SPARQL [1] is a complex event processing system which extends the ETALIS system by a flow-ready extension of SPARQL for query processing [1]. It provides a garbage collection facility which can "prune outdated events" or "expired events by using periodic events, generated by the system". ETALIS is a prolog engine which seriously hampers the applicability of our approach to ETALIS engine. Note however, that EP-SPARQL may be implemented on other engines, e.g., CQUELS, too.

CQELS [10] "implements the required query operators natively to avoid the overhead and limitations of closed system regimes". It optimizes the execution by dynamically re-ordering operators, because "the earlier we prune the triples that will not make it to the final output, the better, since operators will then process fewer triples". This pruning does, however not make any guarantees

⁶ https://jena.apache.org/

about the number of variable bindings created by the processors. Our method should be directly applicable to CQUELS as it provides a native implementation of the operators and these operators maintain a list of active variable bindings.

5.2 Selectivity Estimates

Selectivity estimates for optimizing the execution of SPARQL queries were first investigated by [12]. These are not directly applicable to our approach. Selectivity estimates for SPARQL query optimization base on the selectivity of predicates whereas we are interested in the matching likelihood of variable bindings. More related to our work is the operator scheduling of CQELS [10]. They propose to choose the next operator to execute according to the likelihood that the operator leads to a positive result. They, however, base the decision on heuristics. It would be very interesting to see what effect applying our matching likelihood would have on the performance of CQELS.

To the best of our knowledge, no approach estimates the matching likelihood of bindings to determine their selectivity.

5.3 Load Shedding

Load shedding has been applied to information flow processing. Approaches like [6, 2, 13] perform load shedding by dropping tuples from the stream, i.e., dropping data instead of variable bindings. In contrast to a Complex Event processing (CEP) system they assume a Data Stream Management System, i.e., a set of triples with a relational database execution engine.

An approach we found that follows a similar idea is SASE + [7]. SASE is has an automata based matching approach which can be considered similar to variable bindings in our case. They do apply some eviction strategy. Yet, they base their eviction on a deterministic approach that is similar to our implementation of *fifo*.

Another approach that resembles our work is proposed in [6]. They perform a simple equi-join on two incoming streams and evict tuples which are less likely to find a join partner. However, the approach in [6] works with a sliding window and with a single equi-join of two streams.

Note that applying a window to the flow of RDF triples can be considered dropping tuples, too. As such all approaches that implement a window on the incoming data effectively offer the window as a load shedding strategy.

6 Conclusion and Outlook

In this paper we proposed to perform load shedding by eviction, i.e., by dropping variable bindings rather than by dropping tuples. While the latter is a well-established technique for Data Stream Management Systems, for CEP our approach is the first that applies (i) load shedding by eviction and (ii) bases the decision on the matching likelihood of a variable binding rather than on a heuristic.

Our experiments show that eviction is a promising strategy for regular joins for event-driven Semantic Flow Processing systems. We outperform the usually used deterministic approach first-in-first-out by up to 51% recall.

While our approach is currently investigating recall as the only key-performance indicator (KPI) we are confident that we will be able to show that the matching likelihood also performs better than deterministic approaches when different Quality of Service (QoS) constraints require testing a combination of recall with other KPIs such as response time.

Future work will concentrate on extending the algebra expressions to joins with sequential and temporal constraints. We believe that the solution for load shedding will have to incorporate statistical information from the data flow and the query log. We will also investigate how the matching likelihood can be learned from the stream on-the-go.

In the future we will also plan to evaluate our approach with standardized queries and data as proposed in [14] or [9].

We are convinced that only systematic investigations on the relation between using windows and likelihood-based eviction for load shedding will reveal this solution from the data.

References

- Anicic, D., Fodor, P., Rudolph, S., Stojanovic, N.: EP-SPARQL: a unified language for event processing and stream reasoning. In: Proceedings of the 20th International Conference on World Wide Web. pp. 635–644. ACM (2011)
- Babcock, B., Datar, M., Motwani, R.: Load Shedding for Aggregation Queries over Data Streams. In: Proceedings of the 20th International Conference on Data Engineering. pp. 350—-. ICDE '04, IEEE Computer Society (2004)
- Barbieri, D., Braga, D., Ceri, S.: Incremental reasoning on streams and rich background knowledge. In: Aroyo, L., Antoniou, G., Hyvönen, E., ten Teije, A., Stuckenschmidt, H., Cabral, L., Tudorache, T. (eds.) The Semantic Web: Research and Applications: 7th Extended Semantic Web Conference, ESWC 2010, Heraklion, Crete, Greece, May 30 - June 2, 2010. LNCS, vol. 6088, pp. 1–15 (2010)
- Barbieri, D.F., Braga, D., Ceri, S., Della Valle, E., Grossniklaus, M.: C-SPARQL: A Continuous Query Language for RDF Data Streams. International Journal of Semantic Computing 4(1), 3–25 (2010)
- Cugola, G., Margara, A.: Processing flows of information. ACM Computing Surveys 44(3), 1–62 (Jun 2012)
- Das, A., Gehrke, J., Riedewald, M.: Approximate join processing over data streams. In: Proceedings of the 2003 ACM SIGMOD international conference on on Management of data - SIGMOD '03. p. 40. ACM Press (2003)
- Diao, Y., Immerman, N., Gyllstrom, D.: Sase+: An agile language for kleene closure over event streams. Tech. rep., University of Massachusetts Amherst, Department of Computer Science (2008)
- 8. Harris, S., Seaborne, A.: SPARQL 1.1 Query Language. W3C (2011)
- Le-Phuoc, D., Dao-Tran, M., Pham, M.: Linked stream data processing engines: facts and figures. The Semantic Web-...1380(24761), 1–12 (2012)

- Le-phuoc, D., Dao-tran, M., Parreira, J.X., 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.) The Semantic Web – ISWC 2011: 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I. Lecture Notes in Computer Science, vol. 7031, pp. 370–388. Springer Berlin / Heidelberg (2011)
- Scharrenbach, T., Urbani, J., Margara, A., Valle, E.D., Bernstein, A.: Seven Commandments for Benchmarking Semantic Flow Processing Systems. In: Proc.ESWC 2013 (to appear). pp. 1–15. No. 296126 (2013)
- Stocker, M., Seaborne, A., Bernstein, A., Kiefer, C., Reynolds, D.: SPARQL Basic Graph Pattern Optimization Using Selectivity Estimation. In: Proceedings of the 17th International World Wide Web Conference (WWW). ACM (Apr 2008)
- Tatbul, N., Çetintemel, U., Zdonik, S., Cherniack, M., Stonebraker, M.: Load Shedding in a Data Stream Manager. In: 29th International Conference VLDB. vol. 54, pp. 309–320. VLDB Endowment (2003)
- Zhang, Y., Duc, P.M., Corcho, O., Calbimonte, J.p.: SRBench : A Streaming RDF / SPARQL Benchmark. In: The Semantic Web - ISWC 2012: 11th International Semantic Web Conference, ISWC 2012, Boston, MA, USA, Nov 12-14, 2012, Proceedings (2012)