2018 IEEE 34th International Conference on Data Engineering

Complex Event Processing under Constrained Resources by State-Based Load Shedding

Bo Zhao (expected graduation: March 2020)
Supervised by Matthias Weidlich

Humboldt-Universität zu Berlin, Germany
bo.zhao@hu-berlin.de

***Abstract*—Complex event processing (CEP) systems evaluate queries over event streams for low-latency detection of user-specified event patterns. They need to process streams of growing volume and velocity, while the heterogeneity of event sources yields unpredictable input rates. Against this background, models and algorithms for the optimisation of CEP systems have been proposed in the literature. However, when input rates grow by orders of magnitude during short peak times, exhaustive real-time processing of event streams becomes infeasible. CEP systems shall therefore resort to best-effort query evaluation, which maximises the accuracy of pattern detection while staying within a predefined latency bound. For traditional data stream processing, this is achieved by load shedding that drops some input data without processing it, guided by the estimated importance of particular data entities for the processing accuracy.**

In this work, we argue that such input-based load shedding is not suited for CEP queries in all situations. Unlike for relational stream processing, where the impact of shedding is assessed based on the operator selectivity, the importance of an event for a CEP query is highly dynamic and largely depends on the state of query processing. Depending on the presence of particular partial matches, the impact of dropping a single event can vary drastically. Hence, this PhD project is devoted to state-based load shedding that, instead of dropping input events, discards partial matches to realise best-effort processing under constrained resources. In this paper, we describe the addressed problem in detail, sketch our envisioned solution for state-based load shedding, and present preliminary experimental results that indicate the general feasibility of our approach.

I. Introduction

Devices such as RFID readers, GPS navigators, and smartphones continuously collect and generate data for tracking and monitoring purposes. Hence, we face the challenge of analysing an unprecedented volume of sensed data in (quasi) real-time. In this context, an *event* denotes an instantaneous occurrence of a situation of interest at a particular point in time [6]. Based on this notion, Complex Event Processing (CEP) became the foundation of an emerging class of applications in domains such as finance [20], smart grids [9], urban transportation [5], or supply chain management [21]. CEP systems continuously evaluate queries that correlate the events of an input stream. Query matches then result in a stream of more high-level events, thereby enabling an interpretation of the input stream.

A typical CEP query defines a sequence of event variables, a set of predicates that indicate value correlations, and a time window. Furthermore, it may define the structure of the complex event to generate upon matching a pattern. We illustrate these essential concepts by means of an example.

Example 1: We consider the case of bike-sharing in Beijing, China, where more than 700,000 bikes, equipped with GPS sensors and smart locks, are available to 11 million users [1]. Bikes are rented through a smartphone app and are operated in a free-floating model: Users leave and lock a bike wherever and whenever they finish a ride. Consequently, bikes may be parked in obscure places, making them hard to find. Such situations can be detected by evaluating the following query over a stream of events of user interactions and bike availability:

```
PATTERN SEQ (req a, avail+ b[], unlock c)
WHERE diff(b[i].loc, a.loc) < λ, COUNT(b[])>5,
diff(c.loc, a.loc) > λ, c.UID=a.UID
WITHIN 10min
RETURN warning(a.loc, b[i].loc)
```

This query, formalised in the SASE language [3], [24], detects the following pattern: A user requests a bike and there are more than five bikes parking within distance λ. However, the user then unlocks another bike, more than distance λ away. If such a pattern is detected frequently, the operator shall schedule an inspection of the respective area.

CEP systems shall evaluate queries with low latency. If event streams have a high input rate, query evaluation quickly becomes a performance bottleneck, since the number of partial matches that need to be maintained for query evaluation may be exponential in the number of processed events [25]. This yields an exponential worst-case runtime complexity of common query evaluation algorithms. Note that partial matches need to include the actual payload of the matched events in order to enable the potential generation of a complex event, which also yields an exponential space complexity.

We illustrate this issue in Table I. After processing two events of type 'req', r_1 and r_2, denoting requests at a particular time and location by a specific user, the system maintains two partial matches (Table I, left). Now, processing two events of type 'avail', a_1 and a_2, denoting availability of a specific bike (λ-close to the requests), a CEP system maintains eight partial matches: $\langle r_1 \rangle$, $\langle r_2 \rangle$, $\langle r_1, a_1 \rangle$, $\langle r_1, a_1, a_2 \rangle$, $\langle r_1, a_2 \rangle$, $\langle r_2, a_1 \rangle$, $\langle r_2, a_1, a_2 \rangle$, $\langle r_2, a_2 \rangle$. The next event in the stream would, therefore, be evaluated against eight partial matches, even though only four events have been processed so far.

Due to the complexity of evaluating CEP queries, various optimisations have been proposed, such as delayed construction of partial matches [25], pattern sharing [16],semantic query rewriting [23], and compact encodings of partial matches [26].

2375-026X/18/$31.00 ©2018 IEEE
DOI 10.1109/ICDE.2018.00218

Table I. Partial matches for the query of Example 1.

Stream of 'req' events (time, location, user:)
$r_1 = \langle 1, (x_1, y_1), 5 \rangle$
$r_2 = \langle 8, (x_2, y_2), 6 \rangle$

Stream of 'avail' events (time, location, bike:)
$a_1 = \langle 9, (x_3, y_3), 90 \rangle$
$a_2 = \langle 10, (x_4, y_4), 85 \rangle$

Partial matches of `SEQ(req a)`:

a.ts	a.loc	a.UID
1	(x_1, y_1)	5
8	(x_2, y_2)	6

Partial matches of `SEQ(req a, avail+ b[])`:

a.ts	a.loc	a.UID	b.loc	b.BID
1	(x_1, y_1)	5	(x_3, y_3)	90
1	(x_1, y_1)	5	(x_4, y_4)	85
1	(x_1, y_1)	5	(x_3, y_3) (x_4, y_4)	90 85
8	(x_2, y_2)	6	(x_3, y_3)	90
8	(x_2, y_2)	6	(x_4, y_4)	85
8	(x_2, y_2)	6	(x_3, y_3) (x_4, y_4)	90 85

However, when input rates are volatile and may grow unpredictably by orders of magnitude during short peak times, exhaustive real-time query evaluation becomes infeasible and CEP systems shall resort to best-effort processing. That is, CEP systems shall make most effective use of the available resources and maximise the accuracy of pattern detection, while staying within a predefined latency bound.

For traditional stream processing based on relational operators such as selection, projection, and joins, best-effort processing under constrained resources is achieved by *load shedding* [10], [19], [22]. Some of the input data is dropped without processing it. The selection of data to drop is then guided by an estimation of its importance for the processing accuracy, typically based on the selectivity of relational operators [19]. Yet, traditional data streaming systems differ from CEP systems in queries and execution paradigms. They lack common CEP-specific operators (sequences, negation) where the temporal order of events is of great importance, and their partial evaluation results will not grow exponentially.

In this work, we argue that such input-based load shedding is not well-suited for common CEP queries. The main reason is that, unlike in the case of queries built of relational operators, the importance of a specific event for processing is highly dependent on the current state of the CEP system. Processing a single event may not lead to any (partial or complete) match. Yet, when processing the same event with a different set of currently maintained partial matches, the result may be a large number of matches. As explained above, the number of matches may be exponential in the number of processed events. Hence, a single event may have drastic implications depending on the presence or absence of partial matches. This suggests to approach load shedding based on the state of query processing.

Referring to the example given in Table I, for instance, assume that the system processed r_1, r_2, a_1, which yields four partial matches $\langle r_1 \rangle$, $\langle r_2 \rangle$, $\langle r_1, a_1 \rangle$, and $\langle r_2, a_1 \rangle$. If the system is overloaded, input-based shedding would drop the next event a_2. However, in this situation, dropping some partial matches instead of the input event may be a more beneficial strategy. Partial matches $\langle r_1 \rangle$ and $\langle r_1, a_1 \rangle$ are about to expire due to the time window (10min) and thus, unlikely to yield complete matches. Hence, dropping them would reduce the system load with supposedly little impact on the accuracy of query processing. This may free capacity to process event a_2 and reach a state comprising $\langle r_2 \rangle$, $\langle r_2, a_1 \rangle$, and $\langle r_2, a_1, a_2 \rangle$. Due to the time window, these partial matches are more likely to result in complete matches compared to those dropped earlier.

Against this background, this PhD project aims at developing the foundations of state-based methods for complex event processing under constrained resources. In particular, we strive for load-shedding techniques that optimise the accuracy of query evaluation, once processing becomes lossy due to the CEP system being overloaded. Following the above explanation, our load-shedding mechanism shall be state-based, dropping partial matches instead of input events. A realisation of state-based load shedding requires us to *rank* partial matches, based on their contribution to the overall accuracy of query processing as well as the costs induced by them. Intuitively, partial matches that are less likely to contribute to complete matches, but incur high processing costs, are the first to shed.

We see two major challenges for state-based load shedding. First, the contribution as well as the consumption of resources incurred by a partial match are known only in retrospect, once all events that can potentially lead to further (partial or complete) matches have been processed. As such, state-based load shedding needs to assess the *expected* contribution and resource consumption of a partial match, respectively.

The need to predict how a partial match will develop leads to a second major challenge, which is efficiency of the reasoning. Shedding decisions need to be taken instantaneously. Running a complex forecast procedure to estimate expected contributions and resource consumption for each partial match is infeasible when a CEP system becomes overloaded and may thwart any benefit of load shedding in the first place. The question, therefore, is how to design data structures and algorithms, so that load shedding decisions are taken in constant time.

In the remainder, we elaborate in more detail on these challenges and outline our ideas to address them. Specifically, the next section reviews related work on load shedding in stream processing. The problem of state-based load shedding is formalised in Section III. Subsequently, we report on a first approach to address this problem in Section IV and present preliminary experimental results in Section V. We close with a discussion of the next steps of this PhD project in Section VI.

II. Related Work

Techniques for load-shedding have received considerable attention in the literature on data stream processing, unlike for CEP. Aurora [2] was among the first systems for relational stream processing that employed load shedding to cope with bursty input rates. Moving beyond random shedding strategies that drop arbitrary tuples, Tatbul *et al.* [19] proposed semantic load shedding that discards tuples based on their contribution to the query output, measured by a notion of utility. So called 'drop operators' then discard tuples while maximising utility.

Due to the inherent complexity of joins over data streams, several works considered load shedding for streaming joins. For binary equi-joins, Kang *et al.* [14] showed how to allocate computing resources across two input streams based on arrival rates to maximise the number of output tuples. Load shedding decisions, however, may also be taken based on value distributions of the join attributes [7]. Also, GrubJoin [11]

realises load shedding for multi-way streaming joins based on value distributions of attributes.

Recently, shedding mechanisms that are operator independent and do not assume knowledge about value distributions of tuple attributes have presented. Rivetti *et al.* [18] proposed a load-aware shedding technique for distributed streaming systems, which sheds data based on an estimation of the processing duration of tuples. That enables a targeted optimisation of queueing latencies. Also, for such distributed infrastructures, approaches to manage fairness among several federated stream processing systems, such as THEMIS [13], rely on estimates of how much a tuple contributes to a query result. While not targeting load shedding, FERARI [27] performs load balancing among multiple clouds, reducing inter-cloud communications rather than aiming at best effort query evaluation.

Turning to matching of sequential patterns over streams, Li and Ge [15] recently showed how to learn characteristics of partial matches from data processed in the past in order to prioritise input data for processing. Their focus, however, is on approximate processing, where matches are allowed to deviate from what is specified in a query. Also, the query model neglects correlation predicates in queries (i.e., `WHERE` clauses, see Example 1) and all processing decisions are, again, taken per element of the input stream.

A first take on load shedding for CEP queries as discussed above has been presented by He *et al.* [12]. They argue that load shedding algorithms developed for data stream processing, as reviewed above, are inapplicable for CEP queries. The authors then present an analysis of the theoretical complexity bounds for load shedding for CEP queries and outline shedding algorithms. However, shedding is still input-based, applied per event of the input stream, and grounded in pre-defined weights.

We conclude that various load shedding mechanisms have been proposed, mostly in the field of relational data stream processing. However, all existing approaches are input-based: They discard some elements of the input stream. In this work, we argue that for CEP queries this is not always a suitable approach, since the consequences of dropping an input event are subject to a large variability. A more controlled approach shall realise shedding based on state of query processing, discarding partial matches rather than input events.

III. The Problem of State-based Load Shedding

We model the problem of state-based load-shedding for CEP queries using the following concepts. By S, we denote an infinite event stream comprising discrete events, i.e., data elements that denote a particular occurrence of interest [8]. Events carry timestamps that induce the total order of events in the stream. The finite prefix of such a stream up to a time point t is denoted by $S_{|t}$.

Let Q be a CEP query. For our purposes, the exact query model is not relevant. A query may be based on automata, trees of streaming operators, or be grounded in logic formalisms [4]. However, we assume that Q contains operators that are common to CEP queries, see [25], such as sequences of event variables, negation, Kleene closure, value predicates, and time windows.

Given a stream prefix $S_{|t}$ and a query Q, we write $O_{|t}$ for the output event stream generated by evaluating Q over $S_{|t}$. Put differently, $O_{|t}$ is the sequence of (complex) events generated from the matches of Q over $S_{|t}$. In practice, query evaluation incurs a latency, measured between the time the last event of a match arrived (i.e., the first point in time that the match could have been generated) and the time the match has actually been detected. This latency is typically bounded, meaning that matches that are reported late are no longer useful. We capture this in our model as follows: By $\mu(t)$, we model the latency of query processing at time t, which is given by the latencies observed for the matches that have been generated in some fixed-size measurement interval ending at t.

During the evaluation of Q, a system needs to maintain a set of partial matches. We denote by $R(t)$ the set of these partial matches at a time point t. As discussed above, this set may be subject to exponential growth in the size of the stream prefix $S_{|t}$ that has been processed already.

Load shedding happens when a CEP system is overloaded. Here, we assume that such overload situations are reflected in the observed latency of processing being above a pre-defined threshold θ, i.e., $\mu(t) > \theta$ at some time point t.

In an overload situation, our approach is to work with the state of query processing instead of the input stream. That is, rather than realising load shedding by adapting the stream prefix that is considered for processing, we discard partial matches. This makes the evaluation of the next event of the stream less costly, i.e., it increases the throughput of the system, so that the latency drops below the threshold again.

Load shedding that discards partial matches may have consequences on the output stream. While it cannot introduce false positives (discarding partial matches will never increase the set of generated matches), it may result in false negatives. Some events in the output stream may be missing compared to processing the input stream without load shedding. We capture such errors by means of a difference of stream prefixes, $\delta(O_{|t}, O'_{|t})$, that captures how many events of the stream prefix $O_{|t}$ need to be projected to obtain the stream prefix $O'_{|t}$.

Following this line, state-based load shedding requires us to decide *how many* and *which* partial matches to shed. This selection can be captured as an optimisation problem. If the CEP system is overloaded at time t, a state-based load shedding strategy ρ shall reduce the set of partial matches, $\rho(R(t)) = R'(t)$ with $R'(t) \subset R(t)$, such that the latency is below the threshold again, $\mu(t') < \theta$ for $t' > t$, whereas the resulting loss in the output stream is minimal, $\delta(O_{|t'}, O'_{|t'})$ is minimal for $t' > t$ with $O_{|t'}$ and $O'_{|t'}$ being the output streams generated by Q over $S_{|t'}$ without and with load shedding, respectively.

The above formulation of state-based load shedding provides the view on a *single* overload situation of a system. In practice, after shedding at some time point t, latency may be reduced below the threshold only until some time point t'', i.e., $\mu(t') < \theta$ for $t < t' < t''$, but $\mu(t'') > \theta$. However, the above problem formulation directly extends to such cases. Shedding would be triggered again at time point t'', with the goal still being to minimise the resulting loss in the output stream.

IV. Towards State-based Load Shedding

For state-based load shedding, we need to select how many and which partial matches to shed in an overload situation. In this section, we describe our ideas on how to address the second question, the selection of partial matches, and assume that the number of partial matches to shed is fixed. That is, we strive for a ranking of current partial matches that governs the order in which matches are considered for shedding.

Such a ranking of partial matches considers two aspects per partial match $r \in R(t)$ that is present in the CEP system when shedding is triggered at time point t. First, there is the *contribution* of r to the output event stream in terms of the number of complete matches that are produced in the output stream based on r. This is captured in reference to the remaining time-to-live (TTL) of r, which can be calculated at time t based on the time at which r was created and the time window specified by the respective query. We denote this relative contribution of r by $C_+(r|t)$. Second, we consider the *resource consumption* of r, which refers to the number of partial matches that are derived based on r. Again, this aspect must be incorporated in a relative manner, based on the remaining TTL of the partial match r at time t. We denote the relative resource consumption by $C_-(r|t)$. Intuitively, matches with little contribution, low C_+, and severe costs, high C_-, relative to their TTL, are the best candidates for load shedding.

Apparently, neither C_+ nor C_- are known at the time of load shedding, as they depend on future events in the input stream. Hence, we need to estimate these values, yet conduct such prediction with low computational overhead. We assume that the input event stream shows a reasonable level of regularity in terms of correlation among attributes' value distributions. Such assumption holds true in common real-world applications. For instance, in Example 1, by monitoring correlations of locations among the sequence of bike request, bike available, and bike unlock events, a system may identify that most partial matches related to area *A* never finished as full matches, meaning that respective bikes had been successfully found and unlocked. Therefore, it would be wise to discard partial matches within the particular area *A*. On a more abstract level, we expect that the contribution and resource consumption of current partial matches is close to the one observed for similar partial matches in the past. Our approach, therefore, is to learn respective models that exploit similarities of partial matches.

A. Learning a Contribution Model

The contribution $C_+(r|t)$ of a partial match r at time t, is estimated based on past partial matches that had the same characteristics in terms of attribute values, at the same relative time point. To this end, we maintain a model that tracks hash values of all partial matches' attributes and the number of complete matches generated by these partial matches. With $R_r(t) \subseteq R(t)$ as the set of current partial matches that have the same hash as r, and $M_r(t)$ as the set of complete matches generated based on matches in $R_r(t)$, the contribution is defined as $C_+(r|t) = |M_r(t)|/|R_r(t)|$.

Algorithm 1: Learning C_+

```
Input: Stream prefix S_|t; Partial matches R(t); Hash funct. h; Contribution C+
Output: Updated contribution C+
1 foreach event e in stream prefix S_|t do
2     foreach partial match r ∈ R(t), r having no derived partial matches do
3         if r will yield complete match with e then
4             foreach partial match r' ∈ R(t), r' being derived from r do
5                 get time slice t' for r';
6                 update contribution C+(h(r'|t'));
```

However, maintaining this model for every single time point is expensive, especially with large time windows. Hence, we consider an abstraction that defines the measures for time slices instead of single time points. Accordingly, we calculate C_+ solely for each time slice and the size of these slices becomes a tuning parameter for the accuracy of the prediction model. The intuition of this approach is given in Alg. 1.

B. Learning a Resource Consumption Model

Resource consumption of a partial match r is captured based on the partial matches derived based on r in the remaining TTL. As such, estimating the resource consumption can be approached with a similar approach as described above for the contribution. That is, the estimation for a partial match r is based on the resource consumption observed for partial matches in the past, which had the characteristics as r, i.e., the same hash of attribute values, at the same relative time point. Furthermore, to reduce the computational overhead, we also consider this model solely for time slices instead of individual points in time. The algorithmic approach is similar to Alg. 1. However, we now keep statistics about the number of derived matches, which are updated when a complete match is obtained or a partial match is discarded because of a closing time window.

C. Load Shedding

The above models for contribution and resource consumption enable load shedding, as follows. When a system is overloaded at time t, all current partial matches r are ranked based on a scoring function. The latter is defined as a linear combination of $C_+(r|t)$ and $C_-(r|t)$. Then, the partial matches that receive the lowest scores are subject to shedding. As stated above, we consider scenarios in which a fixed number of partial matches is discarded whenever load shedding is triggered.

V. Preliminary Experimental Results

To show the feasibility of our ideas, we developed a stand-alone prototype and considered a workload of a cluster management scenario. We relied on real-world event streams of the Google Cluster-Usage Traces [17] as well as two monitoring queries. Both queries define a sequence of three event variables, include various value predicates, and operate with a time window of several hours. All results have been obtained on a PC (Intel i7-4790 CPU, 8GB RAM, Ubuntu 16.04).

Table II illustrates the accuracy (with respect to processing without load shedding) and average throughput (averaged over five experiment runs) when running the two evaluation queries with different time windows and applying either our approach to

Table II. Accuracy and throughput (e/sec) of Q1 and Q2

shedding strategy	time window	Q1		Q2	
		accuracy	avg throughput	accuracy	avg throughput
SBLS	3 hours	80.50%	77,936	85.34%	505,631
RBLS	3 hours	73.00%	85,273	72.66%	548,768
SBLS	5 hours	82.31%	53,548	70.95%	498,873
RBLS	5 hours	49.11%	70,002	54.44%	549,195
SBLS	7 hours	86.20%	47,128	64.23%	494,393
RBLS	7 hours	33.77%	71,463	41.04%	566,189

state-based load shedding (SBLS) or random shedding (RBLS) of partial matches. Regardless of the chosen strategy, load shedding affects 20% of the partial matches and is triggered by a latency threshold of 150μs (Q1) and 6μs (Q2).

Clearly, SBLS outperforms RBLS in accuracy and the margin becomes larger as the size of time window increases. The throughput observed for our approach, in turn, is slightly lower than with random shedding, due to the need to maintain the contribution and resource consumption models. These results provide us with evidence that our approach can significantly improve the accuracy of complex event processing under constrained resources, with a minor performance overhead.

Figure 1 further shows the accuracy as a function of the weight of the contribution model in the scoring of partial matches, leaving the weight of the resource consumption model fixed. We observe a non-linear dependency, highlighting a potential for tuning our state-based load shedding approach.

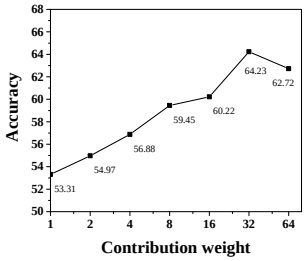

Figure 1: Accuracy vs. weight of contribution model

VI. Discussion and Outlook

We proposed to evaluate CEP queries under constrained resources with state-based instead of input-based load shedding. We outlined our approach to rank partial matches for shedding and presented preliminary experimental results.

As a next step, we will answer the question of how many partial matches to shed, as shedding fixed amounts of data will not yield optimal accuracy. In addition, we intend to come up with more efficient data structures and algorithms, for instance based on sketching, to maintain contribution and resource consumption models, and plan to explore different types of ranking functions and their parametrisations.

Finally, we plan to integrate state-based load shedding into existing optimisations for CEP, e.g., those exploiting semantic information. We expect that knowledge about regularities of a stream helps to achieve an even more effective selection of partial matches for shedding. Also, we aim to explore the integration of our approach with common parallelisation schemes for CEP queries.

References

[1] http://www.washingtonpost.com/world/asia_pacific/china-exports-its-bike-sharing-revolution-to-the-us-and-the-world/2017/08/31/474c822a-87f4-11e7-9ce7-9e175d8953fa_story.html.

[2] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. B. Zdonik. Aurora: a new model and architecture for data stream management. *VLDB J.*, 12(2):120–139, 2003.

[3] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman. Efficient pattern matching over event streams. In *SIGMOD*, pages 147–160, 2008.

[4] A. Artikis, A. Margara, M. Ugarte, S. Vansummeren, and M. Weidlich. Complex event recognition languages: Tutorial. In *DEBS*, pages 7–10. ACM, 2017.

[5] A. Artikis et al. Heterogeneous stream processing and crowdsourcing for urban traffic management. In *EDBT*, pages 712–723. OpenProceedings.org, 2014.

[6] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S. Kim. Composite events for active databases: Semantics, contexts and detection. In *VLDB*, pages 606–617, 1994.

[7] A. Das, J. Gehrke, and M. Riedewald. Approximate join processing over data streams. In *SIGMOD*, pages 40–51, 2003.

[8] O. Etzion and P. Niblett. *Event Processing in Action*. Manning Publications Company, 2010.

[9] R. C. Fernandez, M. Weidlich, P. R. Pietzuch, and A. Gal. Scalable stateful stream processing for smart grids. In *DEBS*, pages 276–281. ACM, 2014.

[10] B. Gedik, K. Wu, and P. S. Yu. Efficient construction of compact shedding filters for data stream processing. In *ICDE*, pages 396–405, 2008.

[11] B. Gedik, K. Wu, P. S. Yu, and L. Liu. A load shedding framework and optimizations for m-way windowed stream joins. In *ICDE*, pages 536–545, 2007.

[12] Y. He, S. Barman, and J. F. Naughton. On load shedding in complex event processing. In *ICDT*, pages 213–224, 2014.

[13] E. Kalyvianaki, M. Fiscato, T. Salonidis, and P. R. Pietzuch. THEMIS: fairness in federated stream processing under overload. In *SIGMOD*, pages 541–553. ACM, 2016.

[14] J. Kang, J. F. Naughton, and S. Viglas. Evaluating window joins over unbounded streams. In *ICDE*, pages 341–352, 2003.

[15] Z. Li and T. Ge. History is a mirror to the future: Best-effort approximate complex event matching with insufficient resources. *PVLDB*, 10(4):397–408, 2016.

[16] M. Ray, C. Lei, and E. A. Rundensteiner. Scalable pattern sharing on event streams. In *SIGMOD*, pages 495–510. ACM, 2016.

[17] C. Reiss, J. Wilkes, and J. Hellerstein. Google cluster-usage traces: format and schema. Technical report, 2013.

[18] N. Rivetti, Y. Busnel, and L. Querzoni. Load-aware shedding in stream processing systems. In *DEBS*, pages 61–68, 2016.

[19] N. Tatbul, U. Çetintemel, S. B. Zdonik, M. Cherniack, and M. Stonebraker. Load shedding in a data stream manager. In *VLDB*, pages 309–320, 2003.

[20] K. Teymourian, M. Rohde, and A. Paschke. Knowledge-based processing of complex stock market events. In *EDBT*, pages 594–597. ACM, 2012.

[21] T. T. L. Tran, C. A. Sutton, R. Cocci, Y. Nie, Y. Diao, and P. J. Shenoy. Probabilistic inference over RFID streams in mobile environments. In *ICDE*, pages 1096–1107. IEEE, 2009.

[22] M. Wei, E. A. Rundensteiner, and M. Mani. Achieving high output quality under limited resources through structure-based spilling in XML streams. *PVLDB*, 3(1):1267–1278, 2010.

[23] M. Weidlich, H. Ziekow, A. Gal, J. Mendling, and M. Weske. Optimizing event pattern matching using business process models. *IEEE Trans. Knowl. Data Eng.*, 26(11):2759–2773, 2014.

[24] E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In *SIGMOD*, pages 407–418, 2006.

[25] H. Zhang, Y. Diao, and N. Immerman. On complexity and optimization of expensive queries in complex event processing. In *SIGMOD*, pages 217–228, 2014.

[26] O. Poppe, C. Lei, S. Ahmed and E. A. Rundensteiner. Complete Event Trend Detection in High-Rate Event Streams. In *SIGMOD*, pages 109–124, 2017.

[27] I. Flouris, *et al.* FERARI: A Prototype for Complex Event Processing over Streaming Multi-cloud Platforms. In *SIGMOD*, pages 2093–2096, 2016.