Load Shedding for Complex Event Processing: Input-based and State-based Techniques

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

Nguyen Quoc Viet Hung
Griffith University, Australia
quocviethung.nguyen@griffith.edu.au

Matthias Weidlich
Humboldt-Universität zu Berlin, Germany
matthias.weidlich@hu-berlin.de

***Abstract*—Complex event processing (CEP) systems that evaluate queries over streams of events may face unpredictable input rates and query selectivities. During short peak times, exhaustive processing is then no longer reasonable, or even infeasible, and systems shall resort to best-effort query evaluation and strive for optimal result quality while staying within a latency bound. In traditional data stream processing, this is achieved by load shedding that discards some stream elements without processing them based on their estimated utility for the query result.**

We argue that such input-based load shedding is not always suitable for CEP queries. It assumes that the utility of each individual element of a stream can be assessed in isolation. For CEP queries, however, this utility may be highly dynamic: Depending on the presence of partial matches, the impact of discarding a single event can vary drastically. In this work, we therefore complement input-based load shedding with a state-based technique that discards partial matches. We introduce a hybrid model that combines both input-based and state-based shedding to achieve high result quality under constrained resources. Our experiments indicate that such hybrid shedding improves the recall by up to 14× for synthetic data and 11.4× for real-world data, compared to baseline approaches.

I. Introduction

Complex event processing (CEP) emerged as a paradigm for the continuous evaluation of queries over streams of event data [14]. By detecting patterns of events, it enables real-time analytics in domains such as finance [39], smart grids [18], or transportation [5]. Various languages for CEP queries have been proposed [14], which show subtle semantic differences and adopt diverse computational models, e.g., automata [2] or operator trees [30]. Yet, they commonly define queries based on operators such as sequencing, correlation conditions over the events' data, and a time window.

CEP systems strive for low latency query evaluation. However, if input rates and query selectivity are high, query evaluation quickly becomes a performance bottleneck. The reason is that query processing is stateful: a CEP system maintains a set of partial matches per query [2]. Unlike for traditional selection and aggregation queries over data streams [4], the state of CEP queries may grow exponentially in the number of processed events and common evaluation algorithms show an exponential worst-case runtime complexity [43].

The inherent complexity of CEP imposes challenges especially in the presence of dynamic workloads. When input rates and query selectivities are volatile, hard to predict, and change by orders of magnitude during short peak times, preallocating sufficient computational resources is no longer reasonable. Permanent overprovisioning of resources to cope with peak demands incurs high costs or is even infeasible. At the same time, scale-out of stream processing infrastructures provides only limited flexibility. For instance, resharding a stream in Amazon Kinesis to double the throughput may take up to an hour if the stream comprises around 100 shards already [3].

Against this background, CEP systems shall employ *best-effort processing*, when resource demands peak [24]. Using resources effectively, the system shall maximize the result quality of pattern detection, while satisfying a latency bound.

Best-effort stream processing may be achieved by *load shedding* [38] that discards some elements of the input stream. Simple strategies that shed events randomly have been implemented for many state-of-the-art infrastructures, e.g., Heron [19], and Kafka [7]. More advanced strategies discard elements based on their estimated utility for traditional selection and aggregation queries [21], [38], [41], [33].

Such input-based load shedding is not always suited for CEP queries, where the utility of a stream element is highly dependent on the current state of query evaluation. Under different sets of partial matches, an event may lead to none or a large number of new matches. Since the number of matches may be exponential in the number of processed events, an event may have drastic implications depending on the presence of certain partial matches. This led to the following observation:

> *"The CEP load shedding problem is significantly different and considerably harder than the problems previously studied in the context of general stream load shedding."*
>
> – Y. He, S. Barman, and J.F. Naughton [24]

This paper argues for a fundamentally new perspective on load shedding for CEP. Since the state of query evaluation governs the complexity of query evaluation, we introduce *state-based load shedding* that discards partial matches, thereby maximizing the recall of query processing in overload situations. Our idea is that state-based shedding offers more fine-granular control, compared to shedding input events. Yet, input-based shedding is generally more efficient: A discarded event is not processed at all, whereas a partial match already incurred some computational effort. When striving for the right balance between result quality and efficiency for a given application, therefore, we need a *hybrid* approach that combines state-based and input-based load shedding. To realize this vision, we need to address several research challenges, as follows:

2375-026X/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDE48307.2020.00099

IEEE computer society

1) The utility of a partial match shall be determined for shedding decisions. This assessment needs to take into account that a partial match may lead to an exponential number of matches, in the number of processed events.
2) State-based and input-based load shedding shall be balanced, which requires establishing a relation between the utilities of input events and partial matches. This is difficult, as a single event may be incorporated in a large number of partial matches, or none at all.
3) Utility assessment and balancing of shedding strategies shall be done efficiently. In overload situations, it is infeasible to run complex forecasting procedures for ranking matches and balancing shedding strategies.

In the light of the above research challenges, this paper makes the following contributions:

- We present *hybrid load shedding* for CEP. It combines input-based load shedding with a fundamentally new approach to shed the state of query processing.
- We propose a *cost model* for hybrid load shedding. It quantifies the utility of partial matches, which is directly linked to the utility of input events.
- Based on this cost model, we develop *efficient decision procedures* for hybrid load shedding. We show how the setting can be formulated as a knapsack problem and how shedding strategies are based on its solution.
- We discuss *implementation considerations* for hybrid load shedding considering the cost model granularity, its estimation and adaptation, and approximation schemes.

We evaluated our approach using synthetic as well as real-world datasets. In comparison to other shedding strategies, given a latency bound, our hybrid approach improves the result quality up to 14× for synthetic data and 11.4× for real-world data.

In the remainder, §II outlines the challenges of load shedding in CEP. In §III, we formalize the load shedding problem for CEP, while the foundations of our approach are detailed in §IV. Practical considerations are discussed in §V. Evaluation results are given in §VI. We close with a discussion of related work (§VII) and conclusions (§VIII).

II. Background

A. Properties of CEP Applications

For a CEP application, an important stream characteristic is *steadiness* of the input rate and the distributions of the events' payload data and, hence, query selectivity. If the rate and distributions are volatile, the size of the state of a CEP system may fluctuate drastically, which is further amplified by the potential exponential growth of partial matches.

Moreover, applications differ in the required guarantees for the *latency* and *quality* of pattern detection. While CEP systems strive for low-latency processing, the precise requirements are application-specific. How the usefulness of query matches deteriorate over time varies greatly, and matches may become completely irrelevant after a certain latency threshold. Yet, depending on the application, it may be acceptable to miss a few matches if the latency for the detected matches is much lower. We illustrate the above properties with example applications:

Urban transportation. Operational management may exploit movements of buses and shared cars or bikes, as well as travel requests and bookings by users [5]. Yet, the resulting streams are unsteady, as, e.g., a large public happening leads to spikes in event rates and query selectivities (many ride requests to a single location). Also, the utility of pattern detection deteriorates quickly over time, whereas the result quality may be compromised for some queries. For instance, queries to correlate requests and offers for shared rides must be processed with sub-second latency to achieve a smooth user experience. Yet, in overload situations, detecting some matches quickly is more profitable than detecting all matches too late or investing into resource overprovisioning.

Example 1: Consider *citibike*, a bike sharing provider that published trip data of 146k members [11]. Bikes are rented through a smartphone app, where users search bikes at nearby stations, start a ride, and finish trips, again at a station. Since bikes quickly accumulate in certain areas, the operator moves around 6k bikes per day among stations. Hence, the detection of 'hot paths' of trips promises to improve operational efficiency.

```
PATTERN SEQ(BikeTrip+ a[], BikeTrip b)
WHERE a[i+1].bike=a[i].bike AND b.end∈{7,8,9}
AND a[last].bike=b.bike AND a[i+1].start=a[i].end
WITHIN 1h
```

Listing 1: Query to detect 'hot paths' of stations.

Listing 1 shows a CEP query to detect such 'hot paths', using the SASE language [2]: Within an hour, a bike is used in several subsequent trips, ending at particular stations. Here, the Kleene closure operator detects arbitrary lengths of paths. Evaluating the query over the citibike dataset [11] reveals a drastic spike in the number of partial matches maintained by the CEP system, see Fig. 1. While higher resource demands may eventually be addressed by scaling out the system, load shedding helps to keep the system operational in peak situations.

Fraud detection. To detect fraud in financial transactions, CEP queries identify suspicious patterns (e.g., a card is suddenly used at various locations). The event streams vary in their input rates and query selectivities, e.g., due to data breaches being exploited. While such variations can hardly be anticipated, there are tight latency bounds for processing: In around 25ms, a credit card transaction needs to be cleared or flagged as fraud [17]. Also, payment models of companies such as Fraugster [20] that penalise false positives make it impossible to simply deny all transactions in sudden overload situations. Hence, a CEP system shall resort to best-effort processing, detecting as many fraudulent transactions as possible within the latency bound.

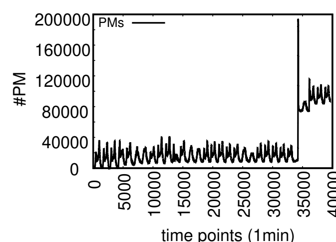


Figure 1: Number of partial matches over time, when evaluating the CEP query given in Listing 1.

B. *Load Shedding for Data Streams*

Load shedding has received attention in the broad field of stream processing. Random shedding strategies have been implemented in current streaming systems, such as Heron [19], and Kafka [7], while shedding may also be guided by queueing latencies [36], concept drift detection [26], and the expected quality of service [33]. Turning to techniques that incorporate the semantics of operators, various strategies have been presented for relational data stream processing. Aurora [1] and Borealis [8] include load shedding functionality, which discards tuples based on their contribution to the query result, measured by a notion of utility [38]. Similar approaches have been presented for relational range queries [21] and XML path queries [41]. Concept-driven [26] monitors the evolution of statistic of queries and streams, and their drift per time window, to determine the utility of input tuples. These approaches require a precise utility estimation per input tuple, whereas for CEP queries, this utility depends on the processing state.

For joins of data streams, which are also stateful operators, load shedding may be based on arrival rates and temporal correlations [25], [22], or value distributions of attributes [23]. Moreover, shedding based on concept-driven detection also enable tuning of a sampling rate per stream [26]. Still, all these approaches define cost models solely on the input stream.

The aforementioned techniques are not applicable for CEP, though [24], as we discuss based on the questions of *when to shed* (Q1); *what to shed* (Q2); and *how much to shed* (Q3).

Existing techniques answer Q1 by relating input rates of streams to processing rates of operators, see [38], [21]. This is infeasible for CEP [24], due to the high volatility of query selectivity and, therefore, processing rates of a system.

Q2 is commonly approached based on the selectivity of relational operators and its changes over time [38], [41], [26]. These approaches exploit, though, that the impact of shedding can be determined rather accurately per stream element. For CEP queries, this is not the case as the utility is state-dependent.

The decision about the amount of data to shed, Q3, is typically governed by the difference of input rates and processing rates [38], [21], [41], [33]. Again, large fluctuations in query selectivity render such an approach unsuitable for CEP systems.

III. Load Shedding in CEP

A. *Event Stream and Query Model*

An event is an instantaneous, unique, and atomic *occurrence of interest* at a point in time. We adopt a tuple-based event model, similar to traditional data stream processing [4]. An event schema is a finite sequence of attributes $A = \langle A_1, \ldots, A_n \rangle$, each being of a primitive data type. An instance of A is an event $e = \langle a_1, \ldots, a_n \rangle$ with a_i being the value of attribute A_i. An event e is assigned a timestamp $e.t$ from a discrete, totally-ordered domain, here defined as $\mathbb{N}$.

A stream $S = \langle e_1, e_2, \ldots \rangle$ is an infinite sequence of events. It is ordered by timestamps, i.e., for any two events e_i and e_j of the stream, $i < j$ implies that $e_i.t \leq e_j.t$. The finite prefix of stream S up to index k is defined as $S(..k) = \langle e_1, \ldots, e_k \rangle$.

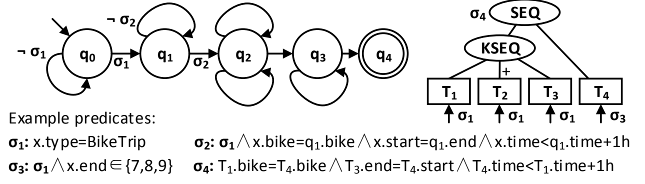


Figure 2: Illustration of computational models.

Common operators of CEP queries include conjunction, sequencing, and Kleene closure [14]. Event selection is further controlled by correlation conditions over the events' payload data and a time window. Most systems adopt similar query languages, but differ in the applied computational models.

For the query of Listing 1, Fig. 2 shows the automata-based model of SASE+ [2] and the tree-based model of ZStream [30]. In the automaton, state transitions are guarded by the query predicates, which correlate the current input event (denoted by x) with events that are part of partial matches (denoted by the state identifier). For instance, `x.bike=q1.bike` in predicate σ_2 means that the current event's bike ID shall be equal to the bike IDs of events of partial matches in state q_1.

In the tree-based model, events are inserted into a hierarchy of input buffers that are guarded by a predicate. Query evaluation proceeds from the leaves to the root, filling operator buffers with event sequences derived from the child buffers.

We largely abstract from the specifics of query languages and computational models. By Q, we denote a query with τ_Q being its time window. Given a stream $S = \langle e_1, e_2, \ldots \rangle$, evaluating Q over S creates complete matches, each being a finite sequence of events $\langle e'_1, \ldots, e'_m \rangle$ of S that preserves the stream order (for $e'_i = e_k$ and $e'_j = e_l$ it holds that $i < j$ implies $k < l$) and satisfies the time window ($e'_m.t - e'_1.t \leq \tau_Q$).

A CEP system evaluates a query Q over a stream S and emits complete matches as soon as they materialize. Query evaluation is incremental and stateful: A stream is processed event by event and a set of partial matches is maintained. These partial matches are sequences of events of S that preserve the stream order and satisfy the query time window. They correspond to partial runs of an automaton or sequences in the buffers at non-root nodes in a tree. As the automaton may be non-deterministic or a tree operator may enumerate all subsequences of a sequence of events, the number of partial matches may be exponential in the size of the processed stream prefix.

Let $S(..k)$ be a prefix of stream S. We write $P(k) = \{\langle e_1, \ldots, e_n \rangle, \ldots, \langle e'_1, \ldots, e'_m \rangle\}$, for the set of partial matches in the CEP system after evaluating query Q over $S(..k)$. Processing of the next stream event, $S(k+1)$, by the CEP system corresponds to a function f_Q of the following signature:

$$f_Q(S(k+1), P(k)) \mapsto P(k+1), C(k+1) \tag{1}$$

The system maps the event $S(k+1)$ and the current partial matches $P(k)$ to a new sets of partial matches $P(k+1)$ and complete matches $C(k+1)$. Query evaluation then yields a stream of sets of complete matches $R = \langle C(1), C(2), \ldots \rangle$. Ordering the complete matches per set and constructing a single event per match, R is transformed into a stream again.

Table I: Overview of notations.

Notation	Explanation
$e = \langle a_1, \ldots, a_n \rangle$	Event
$e.t$	Event timestamp
$S = \langle e_1, e_2, \ldots \rangle$	Event stream
$S(..k)$	Event stream prefix at up to the k-th input event
$P(k)$	Partial matches up to the k-th input event
$C(k)$	Complete matches up to the k-th input event
R	Output stream of complete matches
$\mu(k)$	Query evaluation latency after the k-th input event
θ	Latency bound
ρ_I	Input-based shedding function
ρ_S	State-based shedding function

In the remainder, we focus on queries that are *monotonic*, in the stream and the partial matches. Let $P(k)$ and $P(l)$ be the set of partial matches after evaluating query Q over stream prefix $S(..k)$ and $S(..l)$, $k < l$. A query is monotonic in the stream, if the partial matches $P(k')$ obtained when evaluating Q over an order preserving projection $S'(..k')$ derived by removing some events of $S(..k)$ is a subset of the original ones, $P(k') \subseteq P(k)$. A query is monotonic in the partial matches, if the complete matches $C'(l)$ obtained when evaluating Q over $S(..l)$ using only a subset $P'(k) \subseteq P(k)$ of the partial matches yields a subset of the original complete matches, $C'(l) \subseteq C(l)$. Put differently, for a monotonic query, the removal of input events may only reduce the set of partial matches and the removal of partial matches may only reduce the set of complete matches.

CEP queries that include conjunction, sequencing, Kleene closure, correlation conditions, and time windows are monotonic under an exhaustive event selection policy (e.g., *skip-till-any-match* [2]). The same holds true for common aggregations, such as a query assessing whether the average of attribute values of a sequence of events is larger than a threshold. Intuitively, an exhaustive event selection policy leads to all possible combinations of events and, hence, aggregate values being represented by partial matches derived from the original stream. Therefore, removing an input event or a partial match can only lead to missing complete matches, but will never create further partial or complete matches. Counter-examples for monotonicity are queries with more selective policies, e.g., those that require *strict contiguity* [2] of events, and negation operators. Note though that exhaustive policies represent the most challenging scenario from a computational point of view, so that load shedding is particularly important.

Query evaluation incurs a latency—the time between the arrival of the last event of a complete match and the actual detection. In our model, this corresponds to the time needed to evaluate function f_Q. We denote the latency observed for the complete matches $C(k)$ by $\mu(k)$. In practice, however, latency is assessed for a fixed-size interval, e.g., as a sliding average over 1,000 measurements. To keep the notation concise, we assume that such smoothing is incorporated in $\mu(k)$. Table I summarises our notations.

B. The Load Shedding Problem in CEP

To realize load shedding in a CEP system, we revisit the three questions raised in §II-B: *when* to conduct load shedding (Q1), and *what* (Q2) and *how much* (Q3) data to shed.

The latency of query evaluation, $\mu(k)$, is subject to application-specific requirements (§II-A). We thus consider a model in which load shedding is triggered when the latency $\mu(k)$ exceeds a bound θ (Q1). In practice, the effect of load shedding may materialize only with a minor delay, so that the bound θ shall be chosen slightly smaller than the bound that renders matches irrelevant in the application domain.

Solutions for the decisions of *what* and *how much* to shed (Q2 and Q3) have to consider the quality of query evaluation. We assess this quality as the loss in complete matches induced by shedding. Let $R = \langle C(1), \ldots, C(k) \rangle$ be the results (sets of complete matches) obtained when processing a stream prefix $S(..k)$, and let $R' = \langle C'(1), \ldots, C'(k) \rangle$ be the results obtained when processing the same prefix, but with load shedding. For monotonic queries, it holds that $C'(i) \subseteq C(i)$, $1 \leq i \leq k$, so that $\delta(k) = \sum_{1 \leq i \leq k} |C(i) \setminus C'(i)|$ is the *recall loss*, the total number of complete matches lost due to shedding. Any shedding decision (*what* and *how much*) shall therefore aim at minimizing this loss in recall.

Problem 1: The problem of *load shedding in CEP* is to ensure that when evaluating a CEP query for a stream prefix $S(..k)$, it holds that $\mu(k) \leq \theta$ for $1 \leq k$ and $\delta(k)$ is minimal.

C. Hybrid Shedding Approach

To address the load shedding problem, we propose a fundamentally new perspective on how to decide on *what* (Q2) and *how much* (Q3) data to shed. We introduce *hybrid* load shedding that discards both, input events and partial matches.

Taking up our formalization of query evaluation as a function f_Q that is applied to the next stream event and the current partial matches, i.e., $f_Q(S(k+1), P(k))$, we distinguish *input-based* shedding and *state-based* shedding, formalised by two functions ρ_I and ρ_S:

$$\rho_I(e) \mapsto \begin{cases} e \\ \perp \end{cases} \quad \text{and} \quad \rho_S(P) \mapsto P', \text{ s.t. } P' \subseteq P. \tag{2}$$

That is, ρ_I filters a single event and potentially discards it (denoted by $\perp$), whereas ρ_S filters a set of partial matches, potentially discarding a subset of them. Based thereon, processing of a stream event $S(k+1)$ is represented in our formal model as the application of the evaluation function f_Q to the results of load shedding, i.e., $f_Q(\rho_I(S(k+1)), \rho_S(P(k)))$. Here, we assume that $f_Q(\perp, \rho_S(P(k+1))) \mapsto \rho_S(P(k+1)), \emptyset$, i.e., shedding an input event does not change the maintained partial matches, nor does it generate complete matches.

Fig. 3 links the two shedding strategies to the aforementioned computational models for CEP.

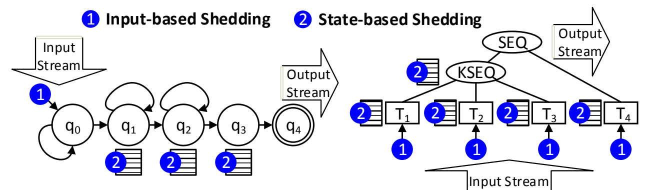


Figure 3: Input-based vs. state-based shedding.

IV. Foundations of Hybrid Shedding

The above approach for hybrid load shedding calls for an instantiation of the input-based and state-based shedding functions. This requires determining the amount of data to shed to ensure that the latency bound is satisfied as well as assessing the utility of input events and partial matches to minimize the loss in recall. To this end, we first introduce a cost model.

A. Cost Model

Input-based and state-based techniques for load shedding differ in the granularity with which the recall and computational effort of query evaluation are affected. Input-based shedding offers coarse-granular control, since a discarded event cannot be part of *any* match. It yields comparatively large savings of computational resources (preventing an exponential number of partial matches), while it may also have a large negative impact on the recall of query evaluation (an exponential number of complete matches may be lost). State-based shedding offers relatively fine-granular control, as the events of a discarded match may remain part of other partial matches. Consequently, the induced computational savings and recall loss are also comparatively small.

The above difference in shedding granularity is important to handle different levels of variance in query selectivity. With small variance, the utility of an input event can be assessed precisely and input-based shedding is preferred: It avoids spending *any* computational resources for processing events with low utility. For a query with a large variance in selectivity, however, an assessment of the utility per event is inherently imprecise, so that resorting to state-based shedding promises higher recall at the expense of smaller resource savings.

To reason on the impact of shedding strategies on the quality of query evaluation and the imposed computational effort, we define a cost model. Striving for a fine-granular assessment, this model is grounded in partial matches. However, it later also serves the selection of input events for shedding.

Consider the moment in time after a stream prefix $S(..k)$ has been processed. At this moment, we assess a partial match along the following dimensions:

Contribution: We assess the *contribution* of a partial match to the query result, i.e., to the construction of complete matches. It is defined by the number of complete matches that are generated by it. With $C(k+1), C(k+2), \dots$ as the complete matches derived in the future, the contribution of a partial match $p = \langle e_1, \dots, e_n \rangle \in P(k)$ is defined as:

$$\Gamma^+(p) = \left| \left\{ \langle e'_1, ..., e'_m \rangle \in C(i) \mid i > k \wedge \forall 1 \leq j \leq n : e'_j = e_j \right\} \right|. \quad (3)$$

Consumption: We assess the *consumption* of computational resources induced by a partial match by considering all partial matches that are derived from it. Unlike for the contribution defined above, we capture the resource consumption explicitly instead of abstracting it by the count of derived matches. The rationale behind is that the resource consumption may vary greatly between partial matches. For instance, the number and complexity of predicates that need to be evaluated for a partial match per input event may differ drastically.

We capture the resource cost of a partial match p by a function $\Omega(p) \mapsto c$, where $c \in \mathbb{N}$. The exact value may be defined as the number of query predicates to evaluate for p (to capture runtime costs) or as its length (to capture the memory footprint). With $P(k+1), P(k+2), \dots$ as the sets of partial matches constructed in the future, the consumption of a partial match $p = \langle e_1, \dots, e_n \rangle \in P(k)$ is defined as:

$$\Gamma^-(p) = \sum_{\substack{\langle e'_1, \dots, e'_m \rangle \in \bigcup_{i>k} P(i) \\ \forall\ 1 \leq j \leq n:\ e'_j = e_j}} \Omega(\langle e'_1, \dots, e'_m \rangle). \quad (4)$$

Contribution and consumption are well-defined, since complete and partial matches obey the time window of a query (see §III-A). This limits the number of matches that can be generated by a single partial match. However, the contribution and consumption of a partial match can only be calculated in retrospect. We therefore later discuss how to construct effective estimators for these measures.

B. Shedding Set Selection

Once the contribution and consumption is known or estimated for partial matches, load shedding based on the following idea. The severity of the violation of the latency bound for query evaluation shall govern the severity of load shedding: The more the bound is violated, the higher the relative share of data that is shed. Specifically, we consider the extent of latency violation as a lower bound for the extent of resource consumption that shall be saved by discarding partial matches.

Consider the situation that shedding has been triggered after a stream prefix $S(..k)$ had been processed. Then, the relative extent of the latency violation is given as $(\mu(k) - \theta)/\mu(k)$. Let $P(k)$ be the set of current partial matches. For each of them, we assess its relative amount of consumed computational resources among all partial matches:

$$\Delta^-(p, P(k)) = \left|\Gamma^-(p)\right| / \sum_{p' \in P(k)} \left|\Gamma^-(p')\right|. \quad (5)$$

Taking the relative extent of latency violation as a lower bound for the relative amount of resource consumption to save, we control the amount of data to shed. That is, we select a subset of partial matches $D \subseteq P(k)$, called a *shedding set*, such that:

$$\sum_{p \in D} \Delta^-(p, P(k)) > \frac{\mu(k) - \theta}{\mu(k)}. \quad (6)$$

While the above formulation provides guidance on the partial matches to consider, shedding shall aim at minimizing the loss in recall of query evaluation (see §III-B). This loss is defined in terms of complete matches missed due to shedding, which links it with our above notion of contribution of a partial match. We therefore assess the relative potential of a partial match $p \in P(k)$ to avoid any loss in recall:

$$\Delta^+(p, P(k)) = \left|\Gamma^+(p)\right| / \sum_{p' \in P(k)} \left|\Gamma^+(p')\right|. \quad (7)$$

Based thereon, we phrase the selection of a shedding set from the set of partial matches as an optimization problem to guide the decisions on what and how much data to shed:

$$\textbf{select} \quad D \subseteq P(k) \quad \textbf{that minimizes} \quad \sum_{p \in D} \Delta^{+}(p, P(k))$$
$$\textbf{subject to} \quad \sum_{p \in D} \Delta^{-}(p, P(k)) > \frac{\mu(k) - \theta}{\mu(k)}. \tag{8}$$

The above problem is a variation of a knapsack problem [27]. Its capacity is defined by the extent of latency violation, which varies among different moments in which load shedding is triggered. Hence, the problem needs to be solved in an online manner. To avoid the respective overheads, we later show how to obtain an approximated solution.

C. Shedding Functions

When load shedding is triggered, a shedding set is computed as detailed above. It is then used to define different shedding strategies by instantiating the functions ρ_I and ρ_S introduced in §III-C for input-based and state-based shedding.

State-based shedding is achieved by not discarding input events and removing all partial matches of the shedding set from the CEP system. Then, ρ_I is the identity function, while ρ_S is defined as $\rho_S(P(k)) \mapsto P(k) \setminus D$. For practical considerations, state-based shedding may not be triggered again immediately, i.e., by the latency $\mu(k+1)$ being above the threshold, but only after some delay $j \in \mathbb{N}$, i.e., by $\mu(k+j)$ the earliest. The intuition is that the effects of shedding first need to materialize, before it is assessed whether further shedding is needed.

Input-based shedding is achieved by not discarding partial matches (ρ_S is the identity function), but deriving the filter ρ_I for input events from the partial matches in the shedding set. Intuitively, the partial matches that are most suitable for load shedding are exploited to derive the conditions based on which input events shall be discarded. Recall that events have a schema, $A = \langle A_1, \ldots, A_n \rangle$, so that each event is an instance $e = \langle a_1, \ldots, a_n \rangle$ of this schema (see §III-A). Given the set of events that are part of matches in the shedding set, defined as $E_D = \{e \mid \exists \langle e'_1, \ldots, e'_m \rangle \in D, 1 \leq i \leq m : e'_i = e\}$, the input-based shedding function is defined as:

$$\rho_I(e) \mapsto \begin{cases} e & \text{if } e \notin E_D, \\ \perp & \text{otherwise.} \end{cases} \tag{9}$$

Input-based shedding by ρ_I applies to the single input event $S(k+1)$ that is to be handled next, after processing the prefix $S(..k)$. Hence, unlike for state-based shedding, to have any effect, input-based shedding needs to be applied for a certain interval. The length of this interval is determined by the latencies $\mu(k+1), \mu(k+2), \ldots$ observed after load shedding was triggered. Once the latency bound is satisfied, $\mu(k+j) \leq \theta$ for some $j \in \mathbb{N}$, input-based shedding is stopped.

Hybrid shedding combines the two above strategies. The shedding set D is used to define function ρ_S to remove partial matches and also serves as the basis for function ρ_I for input-based shedding. Again, the latter function is applied for some interval based on the observed latencies.

A major advantage of hybrid shedding is that it does *not* require explicit balancing of input-based and state-based shedding, e.g., by a fixed weighting scheme. Since both strategies are grounded in the same cost model, balancing is achieved directly by the unified assessment of the consumption and contribution of partial matches and, thus, input events.

V. Implementing Hybrid Shedding

This section reviews aspects to consider when implementing our model for hybrid load shedding.

A. Granularity of the Cost Model

Our cost model for partial matches (§IV-A) is very fine-granular to enable precise shedding decisions. Yet, considering *each* partial match at *any* point in time leads to large computational overhead: The selection of shedding sets (§IV-B) is then based on a knapsack problem with many items, while input-based shedding (§IV-C) becomes costly, due to a potentially complex derivation of input events. We therefore tune the granularity of the cost model through temporal and data abstractions, striving for a balance between the precision of the cost estimation and the computational overhead.

Temporal abstractions: Even though contribution and consumption of matches may change when a single event is processed, there are typically only a few important change points over the lifespan of a partial match. Since exact measurements are not needed for shedding decisions, we employ the temporal abstraction of time slices. The query window, which determines the maximal time-to-live of a partial match, is split into a fixed number of intervals. The cost model is then instantiated per time slice, rather than per time point.

Data abstractions: Partial matches that overlap in their events or the events' attribute values are likely to show similar contribution and consumption values. We therefore lift the cost model to *classes* of partial matches, where each class is characterized by a predicate over the attribute values of the respective events. For instance, in Example 1, partial matches for which the last event denotes a trip ending at stations 3-6 may have similar consumption and contribution values. Assessing costs per class, shedding sets (§IV-B) and shedding functions (§IV-C) are also realized per class. If a class is part of the shedding set, e.g., the function for input-based shedding uses the predicate of the class to decide whether to discard an event.

B. Estimating the Cost Model

To take shedding decisions based on our model, we need to estimate the contribution and consumption of partial matches.

Offline estimation: We evaluate a query over historic data and record partial and complete matches to derive the contribution and consumption of each match. For each partial match, its contribution value is computed by checking how many times its payload was incorporated among complete matches, in the relevant slice of a time window. Its consumption value is computed similarly, by checking against both partial and complete matches.

For each state of the evaluation model (defined by an NFA-state or a buffer in an operator tree), the partial matches are then clustered based on their contribution and consumption values per time slice. Here, clustering algorithms that work with a fixed number of clusters (e.g., K-means) enable direct control of the granularity of the employed data abstraction: Each cluster induces one class for the definition of the cost model. We employ the *gap statistic* technique [40] to estimate an optimal number of clusters. The contribution and consumption per class are computed as the 90th percentiles of the values among the partial matches in the cluster. We keep a data structure that maps the cluster labels to these values.

To use the class estimates in online processing, we need an efficient mechanism to classify a partial match immediately after its creation. We therefore train a classifier for the partial matches of the clusters obtained for each of the states of the computational model, i.e., one classifier per state. The classifier uses the attributes of partial matches that appear in the query predicates as predictor variables. The choice of the classification algorithm is of minor importance, assuming that the classifier can be evaluated efficiently. In this paper, we employ balanced decision trees, setting the maximal depths to the number of clusters for the respective state.

Online adaptation: An instantiation of the cost model based on online clustering and classification is infeasible. However, the estimates per class and time slice may be monitored and adapted. Initially, we start with the classifiers obtained through offline estimation and the mapping of cluster labels to contribution and consumption values. Once a partial match is generated, it is classified using the classifier of the respective state. As a consequence, partial matches are maintained in different classes. However, the contribution and consumption values may change as more events are processed. Therefore, we monitor updates to these values by streaming counts: We maintain the contribution and consumption per class via a lookup table for each state. Upon the creation of a match, the counts for the class and time slice of the originating partial matches are incremented in the lookup table (consumption values). If the new match is a complete match, the counts for contribution values are also incremented. At the end of each time slice, the new contribution value of a class is calculated as $\Gamma^+_{new} = (1-w)\Gamma^+_{old} + w\Gamma^+_{incremented}$. Here, w is the weight of incremented contribution and large values increase the pace of value updating (we set $w = 0.5$). Consumption values are updated following the same procedure. This way, adaptation is based on sketches for efficient streaming counts [13].

C. Approximated Shedding Sets

Selecting a shedding set (§IV-B) requires solving a knapsack problem, which is NP-hard [27]. We found that for a model with tens of classes, computation of shedding sets using dynamic programming [32] takes a few nanoseconds, which is feasible for online processing in overload situations.

If the number of classes is large, approximations shall be applied. For repeated load shedding, shedding sets may be reused, assuming stable contribution and consumption values per class and time slice. Also, the knapsack problem may be approximated, see also [10]. A simple greedy strategy is to select classes of partial matches in the order of their contribution and consumption ratios, until the capacity bound is reached.

Table II: Details on the generated datasets.

	Attribute	Value Distribution
DS1	Type	$\mathcal{U}(\{A, B, C, D\})$
	ID	$\mathcal{U}(1, 10)$
	V	$\mathcal{U}(1, 10)$ (or controlled)
DS2	Type	$\mathcal{U}(\{A, B, C, D\})$
	ID	$\mathcal{U}(1, 10)$
	A.x, A.y, B.x, B.y	$\mathbb{P}(0 < X \leq 2) = 33\%, \mathbb{P}(2 < X \leq 4) = 67\%$
	B.v	$\mathbb{P}(X = 2) = 33\%, \mathbb{P}(X = 5) = 67\%$
	C.v	$\mathbb{P}(X = 3) = 33\%, \mathbb{P}(X = 5) = 67\%$
	D.v	$\mathbb{P}(X = 5) = 33\%, \mathbb{P}(X = 2) = 67\%$

VI. Experimental Evaluation

Below, we first give details on the setup of our evaluation (§VI-A), before turning to the following evaluation questions:

- What are the overall effectiveness and efficiency (§VI-B)?
- How good is the selection of data to shed (§VI-C)?
- How sensitive is the approach to query properties, such as its selectivity, duration, and pattern length (§VI-D)?
- What is the impact of cost model properties (§VI-E)?
- Does the model adapt to changes in the stream (§VI-F)?
- What is the impact of cost model estimation (§VI-G)?
- How are non-monotonic queries handled (§VI-H)?
- How does hybrid load shedding perform for the data of real-world cases (§VI-I and §VI-J)?

A. Experimental Setup

Datasets and queries: For controlled experiments, we generated three datasets as detailed in Table II. Dataset DS1 comprises events with a three-valued, uniformly distributed payload: A categorical type, a numeric ID, and a numeric attribute V. This dataset enables us to evaluate common queries that test for sequences of events of particular types that are correlated by an ID, whereas further conditions may be defined for attribute V. To explore the impact of diverse resource costs of matches (see §IV-A), we generated a second dataset, DS2. The events' payload includes numeric attributes for which values are drawn from partially overlapping ranges.

We execute queries Q1, Q2, and Q4 of Listing 2 over dataset DS1, and query Q3 over dataset DS2. The definition of these queries is motivated by the above evaluation questions. Note that Q1-Q3 are monotonic, whereas Q4 is not (see §III-A). The queries will be explained further in the respective subsections.

We further use the real-world dataset of *citibike* [11], see Example 1. For the trip data of October 2018, we test the query of Listing 1 that checks for 'hot paths'. We configure the query to consider paths of at least five stations, i.e., five is the minimal length of the Kleene closure operator in the query.

As a second real-world dataset, we use the Google Cluster-Usage Traces [35]. The dataset contains events that indicate the lifecycle (e.g., submit (Su), schedule (Sc), evict (Ev), and fail (Fa)) of tasks running in the cluster. We use the query in Listing 3, which detects the following pattern: A task is

submitted, scheduled, and evicted on one machine; later it is rescheduled and evicted on another machine; and finally it is rescheduled on a third machine, but fails execution; within 1h.

Shedding strategies: We compare against several baseline shedding strategies derived from related work:

Random input shedding (RI) discards input events randomly, as implemented, e.g., for Apache Kafka [7].

Selectivity-based input shedding (SI) discards input events by assessing the query selectivity per event type, which corresponds to semantic load shedding as developed for traditional data stream processing with Borealis [8].

Random state shedding (RS) discards partial matches in a purely random manner.

Selectivity-based state shedding (SS) discards partial matches based on the query selectivity for the events in the match, which is inspired by techniques for approximate CEP [29].

For our approach, we test three instantiations of the shedding functions (see §IV-C): *Input-based shedding (HyI)*, *state-based shedding (HyS)*, and *hybrid shedding (Hybrid)*.

We estimate the cost models for 4 time slices and 10 clusters (using K-Means) for each state of queries Q1, Q2, Q4, while 4 clusters per state are derived for query Q3. The number of clusters was used as a maximum depths for the decision tree classifiers learned for each state.

For the citibike scenario, we considered 3 time slices, while the number of clusters in K-Means and the maximal tree depth of the classifiers were set to 15. Turning to the cluster monitoring application, we also relied on 3 time slices. The number of clusters and the maximal tree depth were set to 30.

Measures: We measure the performance of query evaluation under a strict latency bound. In our experiments, this bound is typically defined as a percentage of the latency observed without load shedding. Enforcing the latency bound, we measure the effect of shedding on the result quality and the throughput of the CEP system. Result quality is mostly assessed in terms of recall, i.e., the ratio of complete matches obtained with shedding within the latency bound, and all complete matches, derived without shedding. For monotonic queries, false positives may not occur, so that precision is not compromised. For the non-monotonic query Q4, we also measure precision, though. Throughput is measured in events per second (events/s).

Implementation and environment: We developed an automata-based CEP engine in C++.[1] Following common practice, we rely on indexes over the attribute values of events for efficient evaluation of query predicates. In the same vein, access to matches in the construction of shedding sets is supported by indexes based on the predicates that define the classes of the cost model. The offline estimation of the cost model was parallelized for different sets of partial matches. To reduce the overhead during online adaptation, we further derived lookup tables from the learned classifiers.

Most experiments ran on a workstation with an i7-4790 CPU, 8GB RAM, with Ubuntu 16.04. Cost model estimation took between 0.75 and 4.5 seconds on this machine, which we consider feasible for offline bootstrapping. The results of §VI-J were obtained on a NUMA node with 4 Intel Xeon E7-4880 CPUs (60 cores) and 1TB RAM, running openSUSE 15.0.

[1]Publicly available at *https://github.com/zbjob/AthenaCEP*

```
Q1: PATTERN SEQ(A a,B b,C c)
    WHERE a.ID=b.ID AND a.ID=c.ID AND a.V+b.V=c.V
    WITHIN 8ms
Q2: PATTERN SEQ(A a,A+ b[],B c,C d)
    WHERE a.ID=b[i].ID AND a.ID=c.ID AND a.ID=c.ID
    AND b[i].V=a.V AND a.V+c.V=d.V
    WITHIN 1ms
Q3: PATTERN SEQ(A a,B b,C c,D d)
    WHERE a.ID=b.ID AND a.x≥ b.v/2 AND a.x≤b.v
    AND a.y≥ b.v/2  AND a.y≤b.v AND b.ID=C.ID
    AND c.ID=d.ID AND b.v=d.v AND
    AVG(sqrt((a.x)^2+(a.y)^2)+sqrt((b.x)^2+(b.y)^2))<c.v
    WITHIN 5ms
Q4: PATTERN SEQ(A a, NEG B b,C c)
    WHERE a.ID=b.ID AND a.ID=c.ID
    WITHIN 1ms
```

Listing 2: Queries for experiments with synthetic data.

```
PATTERN SEQ(Su a,Sc b,Ev c,Sc d,Ev e,Sc f,Fa g)
WHERE [task_id] AND b.machine=c.machine
AND b.machine!=d.machine AND d.machine=e.machine
AND d.machine!=f.machine AND f.machine=g.machine
WITHIN 1h
```

Listing 3: Query for Google cluster dataset.

B. Overall Effectiveness and Efficiency

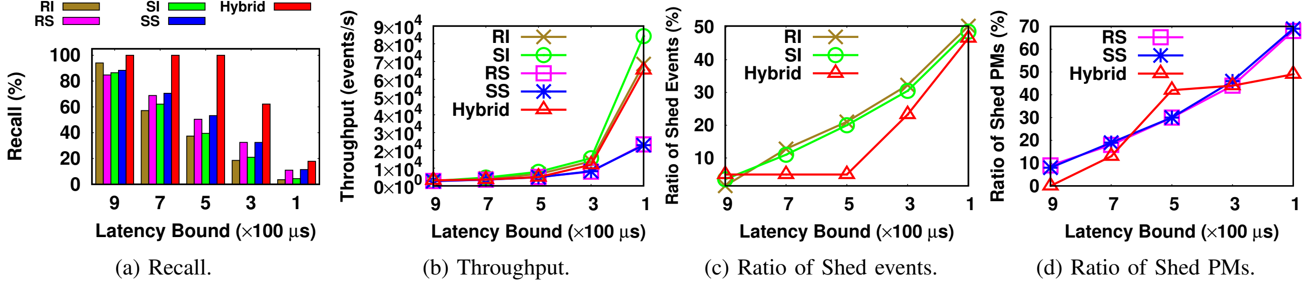
We test the general performance of hybrid load shedding with query Q1 over dataset DS1, drawing attribute V for events of type C from a uniform distribution $\mathcal{U}(2, 10)$. Hence, all events of types A and B may be part of complete matches, but partial matches with $a.V + b.V > 10$ will never lead to a complete match and can thus be discarded without compromising recall.

We test the baseline approaches against our hybrid strategy. Without load shedding, the average latency is 1,033μs, so that we consider bounds between 100μs and 900μs. Fig. 4a shows that hybrid load shedding yields the highest recall. With tighter latency bounds, recall quickly degrades with the baseline strategies, whereas our approach keeps 100% recall for a 900μs - 500μs latency bound. This highlights that our approach is able to assess the utility of partial matches and input events.

State-based strategies yield generally better recall (fine-granular shedding), whereas input-based techniques yield higher throughput (immediate saving of resources), see Fig. 4b. Our hybrid approach is nearly as efficient as the input-based strategies, which is remarkable, given the above recall results.

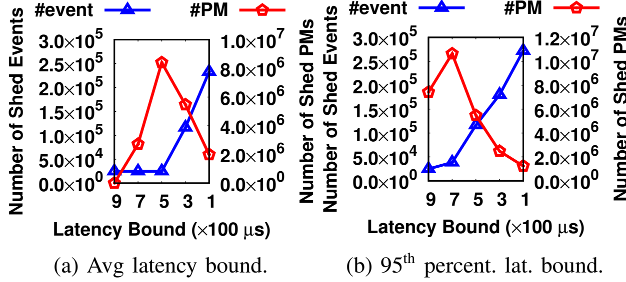
The reason becomes clear when exploring the ratios of shed events and partial matches, Fig. 4c and Fig. 4d. Up to a bound of 500μs, our hybrid strategy discards a steady ratio of input events. The required reduction of latency is achieved by an increasing ratio of shed partial matches, which does not compromise recall (Fig. 4a). Once more input events need to be shed to satisfy the latency bound, the ratio of discarded partial matches flattens. Input-shedding thwarts the generation of partial matches, thereby reducing the shedding pressure.

A repetition of the experiments with the bound being the 95$^{\text{th}}$ percentile latency confirmed the above trends.



(a) Recall. (b) Throughput. (c) Ratio of Shed events. (d) Ratio of Shed PMs.

Figure 4: Experiments when varying the bound enforced for the average latency.



(a) Avg latency bound. (b) 95th percent. lat. bound.

Figure 5: Details on workings of hybrid load shedding.

The above results illustrate that state-based shedding, in general, leads to higher recall. However, throughput is increased more through input-based shedding, since it completely avoids to spend effort on the creation of (potentially irrelevant) partial matches. Hybrid load shedding strives for both, high recall and high throughput, by balancing input-based and state-based shedding. Fig. 5a shows that there is a turning point at the aforementioned latency bound of 500μs, at which the number of shed partial matches decreases and the number of shed input events increases. This behaviour is explained as follows. For tighter bounds, the filter function for input-based shedding (§IV-C) derived from the shedding set (§IV-B) contains more heterogeneous partial matches (i.e., from a larger number of classes and time slices), which increases selectivity of the filter function, while filtering is also applied for longer intervals (until the latency drops below the threshold). Since more input events are filtered, less partial matches are created in the first place, so that the absolute number of shed partial matches also decreases. The result is mirrored for the 95th percentile latency in Fig. 5b, with a turning point at a bound of 700μs.

C. Selection of Data to Shed

For dataset DS1 and query Q1, we assess how well input events or partial matches that do not incur a loss in recall are selected. Fixing the ratio of shed events and matches, Fig. 6a and Fig. 6b show that input-based shedding using our cost model (HyI) yields better recall with slightly worse throughput compared to random (RI) and selectivity-based (SI) input shedding. Hence, our cost model enables a precise assessment of the utility of matches and, thus, events to shed.

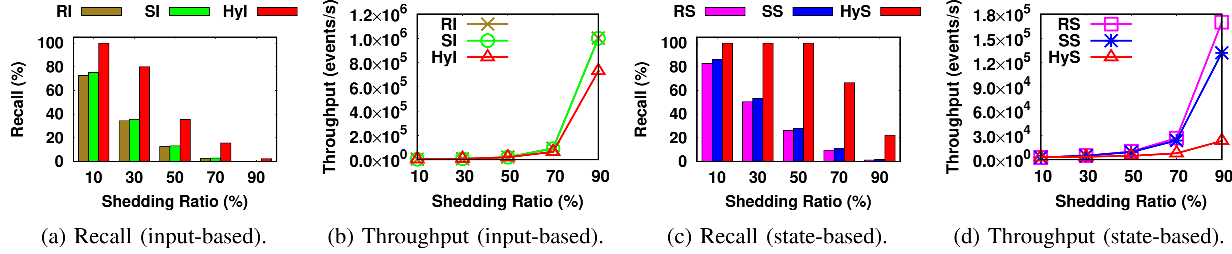
Fig. 6c shows the recall for state-based strategies. Our approach (HyS) shows better recall than random (RS) or selectivity-based strategies (SS). When discarding 50% of the partial matches, our approach keeps 100% recall, whereas the baseline strategies drop to 30%. Interestingly, at that point, all approaches show similar throughput (Fig. 6d). With high shedding ratios, the baselines achieve higher throughput. Yet, this is of little practical value, given the very low recall.

D. Sensitivity to Query Properties

Variance of query selectivity: To test the impact of the variance of query selectivity (Q1 over DS1), we change the distribution of attribute V for C events in $[2, x]$ with $x \in [2, 10]$. This way, we control the overlap of the distributions for A and B events that lead to complete matches. With a 50% bound on the 95th percentile latency, Fig. 7a shows that, as expected, the recall is not affected and hybrid shedding leads to the best results. Fig. 7b, in turn, shows a major impact on throughput. If selectivity shows low variance ($x = 2$), our hybrid approach is able to precisely assess the utility of input events and discard irrelevant ones. Hence, the throughput is 120× higher than the baseline approaches. For high variance, our approach resorts to the more fine-granular level of partial matches, so that the throughput resembles the one of the baseline strategies.

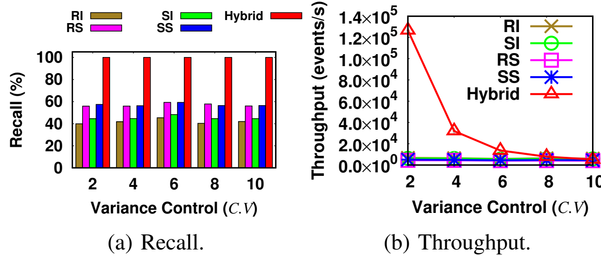
Time window size: Under a steady input rate, the size of a query time window affects the growth of partial matches. We evaluate this effect by varying the window of query Q1 over dataset DS1 from 1ms to 16ms, with a 50% bound on the 95th percentile latency. Fig. 8a shows that our strategy consistently yields the highest recall, while with increasing window size, recall improves for all approaches. This may be attributed to a more precise cost model. Since the number of time slices (§V-A) is kept constant, larger windows mean that more partial and complete matches are used for the estimation. According to Fig. 8b, input-based baseline strategies achieve the best throughput. Our hybrid approach has comparable performance to the state-based strategies. With increasing window size, the differences become marginal due to the exponential growth of the number of partial matches and their increased lifespan.

Pattern length: Using query Q2 over dataset DS1 and a bound for the 95th percentile latency (50%), we vary the limit of the Kleene closure operator to obtain patterns of length four to eight. As shown in Fig. 9a and Fig. 9b, recall remains stable with increasing pattern length, whereas throughput decreases drastically. Interestingly, our approach shows a less severe reduction than the other strategies. Hence, complex queries may particularly benefit from our approach.



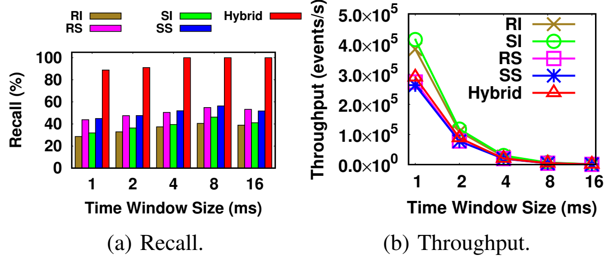
(a) Recall (input-based). (b) Throughput (input-based). (c) Recall (state-based). (d) Throughput (state-based).

Figure 6: Evaluation of the effectiveness of the selection of data to shed.



(a) Recall. (b) Throughput.

Figure 7: Impact of variance of query selectivity.



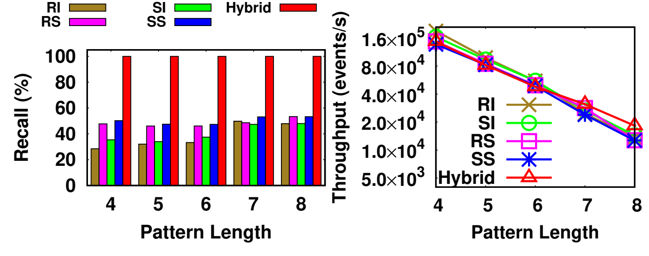
(a) Recall. (b) Throughput.

Figure 8: Impact of time window size.

E. Impact of Cost Model Properties

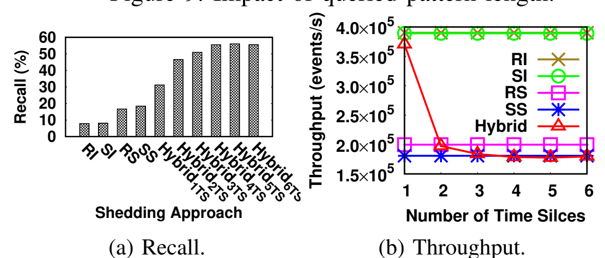
Temporal granularity: We evaluate query Q1 (time window 2ms) over dataset DS1 with a 20% bound on the 95th percentile latency, varying the number of time slices. Fig. 10a depicts the obtained recall, where our hybrid approach is annotated with the number of time slices (TS). While our approach outperforms all baseline strategies, we see evidence for the benefit of using time slices: The highest recall is obtained with $\geq$4 slices. Increasing the number of time slices decreases throughput (Fig. 10b), due to the implied overhead. With a throughput that is on par with RI and SI (one slice), our hybrid approach still yields 3.8$\times$ higher recall. Similar observations are made with respect to the state-based baseline strategies.

Resource costs of partial matches: The consumption of resources may differ among partial matches, which we explore with query Q3 over dataset DS2. The query computes the average Euclidean distance to pairs of numeric values of A and B events, checking whether the result is larger than a value of C events. We established empirically that handling partial matches of A, B events requires 5$\times$ more runtime than handling matches of a single A event. We compare hybrid shedding with and without explicit resource costs for the consumption of partial matches (§IV-A). With a bound on the average latency, Fig. 11a shows that our comprehensive cost model leads to higher recall, at a minor reduction in throughput (Fig. 11b).



(a) Recall. (b) Throughput.

Figure 9: Impact of queried pattern length.



(a) Recall. (b) Throughput.

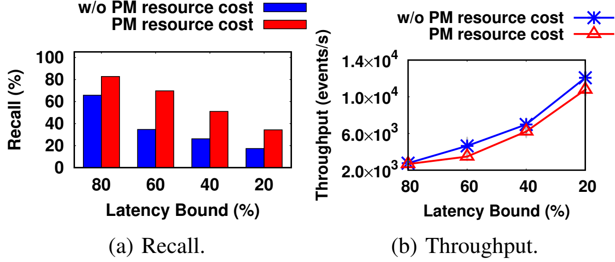
Figure 10: Impact of temporal granularity.

F. Adaptivity to Changes in the Stream

Now, we consider changes in the distributions of the events' payload data. For dataset DS1, we change the distribution of attribute V for C events at a fixed point from $\mathcal{U}(2, 10)$ to $\mathcal{U}(12, 20)$, thereby reversing the costs (worst case setting). With a bound on the average latency (40%), we run Q1 with four time windows (1K, 2K, 4K, and 8K events). Fig. 12 shows how our approach (§V-B) adapts the contribution and consumption estimates: At the change point, recall drops to zero as outdated estimates lead to shedding of all relevant partial matches. However, the change is quickly detected and incorporated. Convergence is quicker for smaller window sizes, due to a shorter lifespan of partial matches.

G. Cost Model Estimation

We evaluate the impact of the cost model estimation with query Q1 over dataset DS1. Q1 has two intermediate states (partial matches of A events and A, B events). We vary the number of clusters from 2 to 10 for each state (max decision tree length is 10). Under a 50% average latency bound, we measure the recall as illustrated in Fig. 13. Overall, the observed recall is not very sensitive to the number of clusters. More clusters lead to higher recall score, but only until reaching a certain number (e.g., 8), after which the effect becomes marginal.



(a) Recall. (b) Throughput.

Figure 11: Impact of resource costs of partial matches.

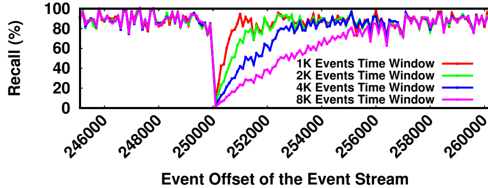


Figure 12: Adaptivity of the cost model.

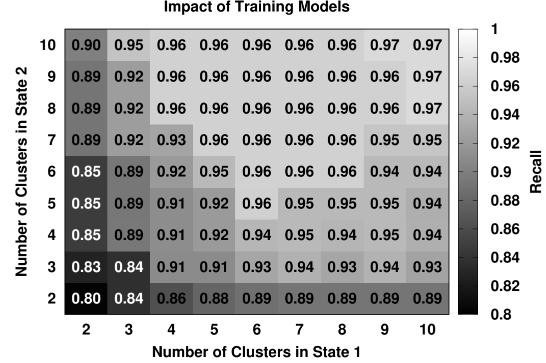


Figure 13: Cost model estimation.

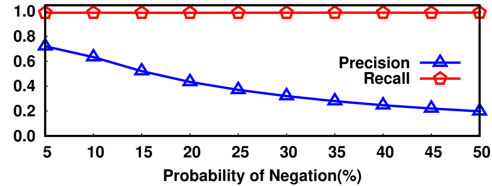


Figure 14: Impact of monotonicity violation.

H. Non-Monotonic Queries

To test the impact of query monotonicity, we rely on query Q4 over dataset DS1. As discussed in §III-A, shedding may produce false positives for non-monotonic queries, so that we measure both precision and recall. We vary the occurrence probability of the negated event type B from 5% to 50%. The other types are evenly distributed. Fig. 14 shows the results when shedding 10% of partial matches. Recall is stable, as our approach discards only the least important partial matches and all complete matches are detected. Yet, precision decreases when increasing the probability of the negation, i.e., the number of false positives becomes larger. Whether this effect is acceptable, depends on the selectivity of the query parts that violate the monotonicity property.

I. Case Study: Bike Sharing

To assess real-world feasibility, we use the *citibike* dataset [11] and the query of Listing 1. We consider all shedding strategies with various bounds on the 99th percentile latency. Here, the selectivity-based approaches (SI, SS) exploit the user type. Our hybrid approach consistently yields the best recall, see Fig. 15a, with the margin becoming larger for tighter latency bounds. At a 20% bound, the recall of our approach reaches 11.4×, 11×, 3.9×, 2.7× the recall of RI, SI, RS, SS, respectively. Fig. 15b shows that the throughput of our hybrid approach is comparable to the state-based strategies (RS and SS), but lower than the input-based strategies (RI and SI). The reason being that, for this dataset, our approach turns out to shed more partial matches than input events.

J. Case Study: Cluster Monitoring

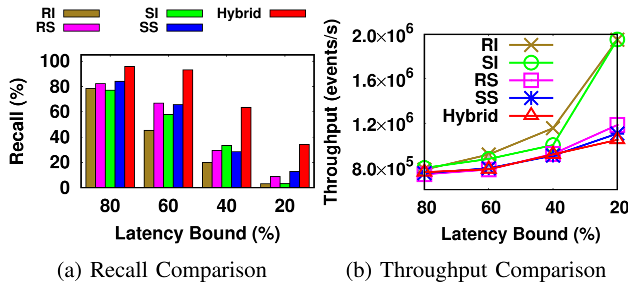
For the Google Cluster-Usage Traces [35], we ran the query in Listing 3 under different latency bounds. Fig. 16a illustrates that hybrid shedding yields the best recall, up to 4× better than with input-based shedding (RI, SI) and 1.5× better than with state-based shedding (RS, SS). Fig. 16b shows the observed throughput, hinting at the general trade-off of input-based and state-based shedding. The former tend to achieve higher throughput at the expense of lower recall. Our hybrid approach achieves similar throughput as the best performing baseline strategy (SI), being slightly slower only for the 20% latency bound. However, hybrid shedding achieves much higher recall, thereby confirming our earlier observations.

VII. Related Work

Since we reviewed related work on load shedding for data streams already in §II-B, this section focuses on techniques for efficient CEP and approximate query processing.

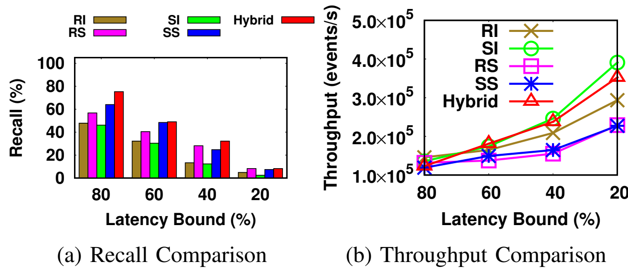
Efficient CEP. The inherent complexity of evaluating CEP queries is widely acknowledged [43] and related optimizations include parallelization [6], sharing of partial matches [43], [34], semantic query rewriting [16], [42], and efficient rollback-recovery in distributed CEP [28]. The characteristics and the complexity of load shedding for CEP has been discussed in [24]. The presented algorithms, however, are limited to input-based shedding and optimize shedding decisions for a set of queries based on pre-defined weights. eSPICE [37] employs the event types' relative positions in a time window to assess the utility of an event. These contribution are largely orthogonal to our work, which optimizes the accuracy for a single query by hybrid load shedding. While we sketched the idea of state-based shedding in [44], this paper presents an operationalization of this idea.

Approximate query processing (AQP). AQP estimates the result of queries [9], e.g., based on sampling or workload knowledge [31]. For aggregation queries, sketches [12] may be employed for efficient, but lossy data stream processing. Recently, AQP was explored for sequential pattern matching [29], with a focus on matches that deviate slightly from what is specified in a query. We took up the idea to learn characteristics from historic data to prioritize data for processing in our baseline that assesses partial matches based on query selectivity. Yet, hybrid shedding outperforms this strategy.



(a) Recall Comparison (b) Throughput Comparison

Figure 15: Case study: Bike sharing.



(a) Recall Comparison (b) Throughput Comparison

Figure 16: Case study: Cluster monitoring.

VIII. Conclusions

In this paper, we proposed hybrid load shedding for complex event processing. It enables best-effort query evaluation, striving for maximal accuracy while staying within a latency bound. Since the utility of an event in a stream may be highly dynamic, we complemented traditional input-based shedding with a novel perspective: shedding of partial matches. We presented a cost model to balance various shedding strategies and decide on what and how much data to shed. Our experiments highlight the effectiveness and efficiency of our approach.

References

[1] D. J. Abadi, D. Carney, U. Çetintemel, et al. Aurora: a new model and architecture for data stream management. *VLDB J.*, 12(2):120–139, 2003.

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

[3] Amazon Kinese. Amazon Kinese Data Streaming FAQs. https://aws.amazon.com/kinesis/data-streams/faqs/, 2019. Last access: 14/10/19.

[4] A. Arasu, B. Babcock, S. Babu, et al. STREAM: the stanford data stream management system. *Data Stream Management*, 317–336, 2016.

[5] A. Artikis, M. Weidlich, F. Schnitzler, I. Boutsis, et al. Heterogeneous stream processing and crowdsourcing for urban traffic management. *EDBT*, 712–723, 2014.

[6] C. Balkesen, N. Dindar, M. Wetter, and N. Tatbul. RIP: run-based intra-query parallelism for scalable complex event processing. *DEBS*, 3–14, 2013.

[7] J. Bang, S. Son, H. Kim, Y. Moon, and M. Choi. Design and implementation of a load shedding engine for solving starvation problems in apache kafka. *IEEE/IFIP*, 1–4, 2018.

[8] U. Çetintemel, D. J. Abadi, Y. Ahmad, et al. The aurora and borealis stream processing engines. *Data Stream Management*, 337–359.

[9] S. Chaudhuri, B. Ding, and S. Kandula. Approximate query processing: No silver bullet. *SIGMOD*, 511–519, 2017.

[10] C. Chekuri and S. Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM J. Comp.*, 35(3):713–728, 2005.

[11] Citi Bike. System Data. http://www.citibikenyc.com/system-data, 2019. Last access: 14/10/19.

[12] G. Cormode, M. N. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4(1-3):1–294, 2012.

[13] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algo.*, 55(1):58–75, 2005.

[14] G. Cugola and A. Margara. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3):15:1–15:62, 2012.

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

[16] L. Ding, K. Works, and E. A. Rundensteiner. Semantic stream query optimization exploiting dynamic metadata. *ICDE*, 111–122, 2011.

[17] feedzai.com. Modern Payment Fraud Prevention at Big Data Scale. http://tiny.cc/feedzai 2013. Last access: 14/10/19.

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

[19] A. Floratou, A. Agrawal, B. Graham, et al. Dhalion: Self-regulating stream processing in heron. *PVLDB*, 10(12):1825–1836, 2017.

[20] Fraugster. https://fraugster.com/, 2019.

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

[22] B. Gedik, K. Wu, P. S. Yu, and L. Liu. Adaptive load shedding for windowed stream joins. *CIKM*, 171–178, 2005.

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

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

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

[26] N. R. Katsipoulakis, A. Labrinidis, and P. K. Chrysanthis. Concept-driven load shedding: Reducing size and error of voluminous and variable data streams. *IEEE Big Data*, 2018.

[27] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack problems*. Springer, 2004.

[28] G.F. Lima, M. Endler, A. Slo, et al. Skipping Unused Events to Speed Up Rollback-Recovery in Distributed Data-Parallel CEP. *BDCAT*, 31–40, 2018.

[29] 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.

[30] Y. Mei and S. Madden. Zstream: a cost-based query processor for adaptively detecting composite events. *SIGMOD*, 193–206, 2009.

[31] Y. Park, B. Mozafari, J. Sorenson, and J. Wang. Verdictdb: Universalizing approximate query processing. *SIGMOD*, 1461–1476, 2018.

[32] U. Pferschy. Dynamic programming revisited: Improving knapsack algorithms. *Computing*, 63(4):419–430, 1999.

[33] T. N. Pham, P. K. Chrysanthis, and A. Labrinidis. Avoiding class warfare: Managing continuous queries with differentiated classes of service. *The VLDB Journal*, 25(2):197–221, 2016.

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

[35] C. Reiss, J. Wilkes, and J. L. Hellerstein. Google cluster-usage traces: format + schema. https://github.com/google/cluster-data.

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

[37] A. Slo, S. Bhowmik and K. Rothermel eSPICE: Probabilistic Load Shedding from Input Event Streams in Complex Event Processing. *Middleware*, 215–227, 2019.

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

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

[40] R. Tibshirani, G. Walther, and T. Hastie. Estimating the number of clusters in a data set via the gap statistic. *J. R. Statist. Soc. B*, 63(2):411–423, 2001.

[41] 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.

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

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

[44] B. Zhao. Complex event processing under constrained resources by state-based load shedding. *ICDE*, 1699–1703, 2018.