Issues in Complex Event Processing: Status and Prospects in the Big Data Era

Ioannis Flouris, Nikos Giatrakos*, Antonios Deligiannakis, Minos Garofalakis, Michael Kamp, Michael Mock

[a] *School of Electronic and Computer Engineering, Technical University of Crete, University Campus., GR-73100 Chania, Greece*
[b] *Fraunhofer IAIS, Schloss Birlinghoven, 53754 St. Augustin, Germany*

Abstract

Many Big Data technologies were built to enable the processing of human generated data, setting aside the enormous amount of data generated from Machine-to-Machine (M2M) interactions and Internet-of-Things (IoT) platforms. Such interactions create real-time data streams that are much more structured, often in the form of series of event occurrences. In this paper, we provide an overview on the main research issues confronted by existing Complex Event Processing (CEP) techniques, with an emphasis on query optimization aspects. Our study expands on both deterministic and probabilistic event models and spans from centralized to distributed network settings. In that, we cover a wide range of approaches in the CEP domain and review the current status of techniques that tackle efficient query processing. These techniques serve as a starting point for developing Big Data oriented CEP applications. Therefore, we further study the issues that arise upon trying to apply those techniques over Big Data enabling technologies, as is the case with cloud platforms. Furthermore, we expand on the synergies among Predictive Analytics and CEP with an emphasis on scalability and elasticity considerations in cloud platforms with potentially dispersed resource pools.

Keywords: Complex Event Processing, Cloud Computing, Predictive Analytics.

1. Introduction

Big Data is generally characterized by four main aspects: Volume, which is the enormous amount of data to be processed; Velocity, which is the speed at which data must be processed; Variety, which represents the multiple representations in the data model; and Veracity, which is the uncertainty in the data. Many modern Big Data applications aim to enhance the processing of human generated data which are though

*Corresponding author. Tel.: (+30) 28210 37265; Fax: (+30) 28210 37542
Email addresses: gflouris@softnet.tuc.gr (Ioannis Flouris), ngiatrakos@softnet.tuc.gr (Nikos Giatrakos), adeli@softnet.tuc.gr (Antonios Deligiannakis), minos@softnet.tuc.gr (Minos Garofalakis), michael.kamp@iais.fhg.de (Michael Kamp), michael.mock@iais.fhg.de (Michael Mock)

surpassed in volume by data produced during Machine-to-Machine (M2M) interactions. M2M data are generated in high frequency in every Big Data system and include useful information that can be utilized to identify the occurrence of interesting situations. Besides M2M interactions, Internet-of-Things (IoT) platforms offer advanced connectivity of devices and services that covers a variety of domains and applications generating voluminous data streams and patterns of interest subjected to further study.

Complex Event Processing (CEP) systems aim at processing such data efficiently and immediately recognize interesting situations when they occur. Generally, events can be thought of as single occurrences of interest in time and complex events as situations or patterns that comprise a particular composite meaning for the system. Each event is assigned an event type based on its semantics and content. Primitive events are atomic (i.e., non-decomposable) occurrences of interest that stream into a CEP system, while composite (or complex) events are detected and extracted by the CEP system based on defined patterns (rules) engaging other primitive and/or other complex event combinations. For instance, imagine a number of sensors measuring the conditions on a production machine. A sensor reporting abnormal vibrations produces a primitive event tuple v of type V. Similarly, a sensor measuring exceptionally high levels of pressure on the same machine yields a primitive event tuple p of type P. Domain experts may have defined a rule stating that *"abnormal vibrations combined with high pressure signal an imminent break down"*. The latter is a complex event, derived by a pattern of the form: AND(V,P).

The previous conceptualization of events and patterns is interdisciplinary enough for modeling a wide range of systems that operate under real-time constraints:

– Data Stream Management Systems (DSMSs) [1, 2] are devoted to processing unbounded streams of data as they arrive and provide real-time answers to continuous or ad-hoc user queries. A DSMS can be transformed to a CEP system by appropriate pipelining answers of continuous queries [3]. However, contrary to DSMSs that leave to their clients the responsibility of attaching a particular meaning to the data being processed, CEP systems encompass the ability to query for complex patterns through predefined rules that match incoming event notifications on the basis of their *content* and on some *ordering relationships* on them [4].

– Time-series data stemming from various information sources can be abstracted as primitive events. For instance, CEP can be used in credit card fraud detection by monitoring credit card activity, i.e., primitive events correspond to transactions, in real-time. It can then perform time-series analysis over streams of events, and it can even correlate the incoming tuples with historical event data like customer information from a CRM system.

– Event types group tuples of similar semantics and content. Different event types can be produced at different sources and thus each event type forms its own event stream. Individual streams flow together in the CEP system synthesizing an heterogeneous stream of tuples. The detection of a pattern expressing a complex event combines information about events of different types. Therefore, CEP is also closely related to real-time knowledge extraction from voluminous cross-stream data.

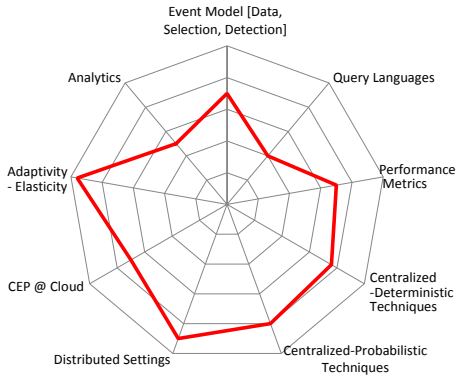

Figure 1: Dimensions & Depth of our Analysis

Real world applications where streams of Big Event Data are of the essence include, but are not limited to, network health monitoring applications, mobile and sensor networks, computer clusters, smart energy grids and security attack detection. In the business sector, accounting, logistics, warehousing and stock trading applications are included among others. Even beyond M2M synergies, human interactions in modern social media enable the potential for event identification based on content of interest hidden, for instance, in Twitter microblogs [5]. More related to human interactions with IoT platforms, the advertising industry can gain much from CEP by successfully implementing IoT-centric solutions, such as modern Internet Advertising [6] approaches, as part of event data-collecting initiatives.

In this paper, we aim to unravel the main research issues tackled in the literature of CEP systems and present the contribution of developed techniques. Different from previous efforts [4, 7, 8, 9], our focal point, without neglecting other query processing aspects, lies on query optimization issues. We treat the deterministic as well as probabilistic event processing model in a unified way and expand our study to different architectural settings, from centralized to distributed networks. In that, we manage to cover a wide range of research efforts in the CEP domain and review the current status of techniques that tackle efficient query processing. Going one step further, we study the issues that arise upon trying to extend those techniques to operate over Big Data enabling technologies and in particular the cloud computing paradigm where query optimization is coupled with elastic resource allocation. Additionally, we examine the synergies among Predictive Analytics (PA) and CEP with an emphasis on scalability and elasticity in cloud platforms with potentially dispersed resource pools. Overall, our work reviews the status of CEP approaches for efficient query processing and points out the prospects and open issues for CEP in the Big Data Era.

Figure 1 illustrates the main dimensions of our upcoming analysis. The distance of the colored line from the center of the graph is proportional to the depth of our discussion compared to the abundance of related work for each dimension and our notion of significance. Details on the content of each axis will be given in the sections to come in a clockwise order. In particular, in Section 2 we present fundamental concepts around CEP approaches. Section 3 reviews centralized CEP approaches in the deterministic

Abbreviation	Meaning
AIG	Active Instance Graph
AIS	Active Instance Stack
BN	Bayesian Network
CEP	Complex Event Processing
DAG	Directed Acyclic Graph
EPN	Event Processing Network
FSM	Finite State Machine
NFA	Non-deterministic Finite Automaton
PA	Predictive Analytics
PAIS	Partitioned [10]/Probabilistic [11] Active Instance Stack

Table 1: Frequently Used Abbreviations

event context, while Section 4 studies probabilistic approaches in the centralized setting. Techniques focusing on complex event processing and monitoring in distributed settings, composed of geographically dispersed processing nodes, are reviewed in Section 5. Section 6 is devoted to issues arising upon operating on Big Data enabling technologies as is the case with the cloud computing paradigm. Eventually, Section 7 discusses the prospects in coupling CEP with PA, before presenting our concluding remarks and outline open research issues in Section 8. Table 1 summarizes the most frequent abbreviations used throughout our study.

2. Fundamental Concepts

2.1. Event Data Stream Model

2.1.1. Deterministic Event Model

A generic event definition would refer to an event as a tuple (occurrence) of interest composed of a number of content attributes denoting an event's meaning, along with one or more time attributes denoting the event's time occurrence and/or its duration. The conceptualization that is most closely related to this abstract event definition is that of [12] which defines events as tuples $e =< s, t >$, where e represents the event of interest, s refers to a list of content attributes and t is a list of timestamps, the first being the start of the event and the last the end of it.

Most research efforts [12], [10], [13], [14] categorize events as primitive and composite (or complex or deferred). Primitive events constitute atomic (non-decomposable) occurrences of interest that stream into a CEP system from monitored data sources. On the other hand, composite (or complex) events are detected and extracted by the CEP system based on defined patterns engaging other primitive and/or other complex event combinations. These patterns correspond to rules, usually defined by domain experts [15], according to which a specified combination of primitive and/or composite event occurrences satisfy the conditions for another composite event apparition.

Both primitive or composite event tuples are grouped into event types (or classes) sharing common semantics or context in their attribute (s) and timestamp lists (t). Equivalently, an event tuple constitutes an instantiation of the event type it belongs to.

2.1.2. Probabilistic/Uncertain Event Model

Despite the fact that the $e =< s, t >$ representation is generic enough to fully describe a deterministic event, the sources that produce the event entities that stream into the CEP system may incorporate imprecisions [7, 8] to either the *event content* (attributes) [16, 17, 15, 18, 19] or even provide faulty judgment about an *event occurrence* [20, 11]. Additionally, the rules (patterns) defining composite events may be probabilistic when applications define that the same set of events participating in a pattern can lead to different composite events with a certain probability for each of the alternatives, i.e., *uncertain rules* [17, 21, 15, 22] may be defined.

In our study we are going to use the terms probabilistic or uncertain for events abiding by the model discussed in this section, interchangeably. Previous works including [16, 17, 15, 18, 19] take into consideration event tuple representations admitting uncertainty on the *event content*. In this case, an attribute belonging to the s or t list is accompanied by its probability density function (pdf). Embodying attribute uncertainty in $e =< s, t >$ can be accomplished in two ways: (a) encapsulate the attribute's pdf together with the attribute's value in the s list, i.e.: $e =< s = [\{attr_1 = val_1, pdf_1(\mu_1, \sigma_1)\}, \ldots, \{attr_d = val_d, pdf_d(\mu_d, \sigma_d)\}], t >$, (b) a much more popular, alternative is to interpret the available information (pdf) about a continuous attribute's value to categorical attribute values, along with their respective probabilities.

In case of uncertain event occurrence, e is represented by a tuple $< e, p_e >$ where p_e denotes its occurrence probability. The previously mentioned works, with the exception of [19], assume or explicitly present (e.g. see [15]) ways so that the uncertainty present in the attribute values of an event may be mapped to an uncertainty value for the event apparition. Hence, they are able to accommodate both types of uncertainty in their frameworks. As regards uncertain rules, the uncertainty value expresses the probability of a complex event occurrence instead of some other alternative. Therefore, the $< e, p_e >$ representation is also capable of encompassing this type of uncertainty.

2.2. Event Query Elements

Queries submitted to a CEP system aim at detecting complex events on the incoming stream of primitive events based on a given pattern. Although, different CEP approaches such as [23, 16, 22] often utilize alternative query elements targeting on their specific event modeling purposes, SASE's [10, 24, 25, 19] query language provides a comprehensive extension of SQL tailored for expressing event queries. The compartments of the following SASE+ based query are indicative of the basic elements of any event detection query:

```
PATTERN ⊕_{i=1}^{n} OPERATORı(list of EventTypes)
WHERE (Qualifications)
WITHIN (time)
[HAVING (Confidence_Qualification)]
RETURN (ComplexEvent or Events to return)
```

The prominent operators in the PATTERN OPERATOR clause are {SEQ, AND, OR, NOT, Kleene Closure} each of them receiving specific event types as its input, while the $\oplus$ symbol expresses any possible combination or nesting of these operators. In

particular the SEQ pattern requires that all events occur *sequentially*, the AND operator declares conjunction of *all* events included in the pattern, OR corresponds to the occurrence of *any* event, while the Kleene Closure operator means *zero or more* individual occurrences for an event included in the pattern. Equivalent expressions of these operators can also be found in the literature, where the SEQ operator is expressed as *sequence* or *next*, the AND operator as *conjunction* or *all*, the OR operator as *disjunction* or *any* and the NOT operator as *negation* or *absence*. The WHERE clause is for one or more (in)equality constraints among event attributes connected by logical operators. The WITHIN clause specifies the time frame (window) in which all events must occur for a full pattern match and thus, a valid composite event detection. The HAVING clause refers to the uncertain event model as described in Section 2.1.2 and specifies a confidence threshold below which primitive event tuples are filtered away or composite event tuples are pruned from further processing [19, 18, 11]. We are going to elaborate more on the utility of this query compartment in Section 4.1. The RETURN clause outputs a complex event and/or its list of primitive events.

2.3. Event Selection Strategies

An event selection strategy determines how the query operators will be applied on a stream of events. Such a strategy may, for instance, dictate that a SEQ operator must select strictly contiguous events of the incoming stream or a less rigid strategy may allow the selection of contiguous "relevant" events, i.e., intermediate events that are not input to the query operator are completely overlooked. Event selection strategy is a feature not widely adopted in the literature, but adds a certain functionality in the detection of complex events. The approach in [26] uses different operators that encapsulate that functionality. A more elaborate approach presented in [25] and in [24] identifies four distinct event selection strategies.

- **Strict Contiguity** requires two selected events to be contiguous in the input stream. This requirement is typical in regular expression matching against strings, DNA sequences etc.
- **Partition Contiguity** is a relaxation of the above strategy. Provided events are conceptually partitioned based on a condition, partition contiguity requires the next relevant event must be contiguous to the previous one in the same partition.
- **Skip till next match** is a further relaxation to remove contiguity requirements. All irrelevant events will be skipped until the next relevant event is read. This strategy is important in many real-world scenarios where some events are "semantic noise" to a particular pattern and should be ignored to enable the pattern matching to proceed.
- **Skip till any match** relaxes the latter strategy by further allowing non-determinism on relevant events. This strategy essentially computes transitive closure over relevant events as they arrive in the system.

In the literature, the default selection strategy is *skip till next match*, while [19] shows that in setups with imprecise/uncertain timestamps event processing must be configured with *skip till any match*, irrespectively of the event selection strategy that is actually dictated by the query.

Approach	Event Detection Model				
	NFA	**FSM**	**EPN**	**Graph**	**Tree**
Moller et al [12]		✓			
SASE [10, 24, 25, 19]	✓				
Adkere et al [13]		✓		✓	
ZStream [14]					✓
Kolchinsky et al [27]	✓				✓
Wasserkrug et al [22, 15, 21]				✓	
Rabinovich et al [28]			✓		
Lahar [16]	✓				
Chuanfei et al [20]	✓				
Shen at al [18]	✓				
Wang et al [11]	✓		✓		
CEP2U [17, 29]	✓				

Table 2: Event Detection Model Representation per Approach

2.4. Event Detection Model Representation

As streaming events arrive to a CEP system, a query pattern is progressively evaluated. This means that before composite event detection via a full pattern match, partial matches of the query pattern take place. These partial matches need to be monitored as they express the potential for an imminent full match. To track the state of a partial match and determine on a complex event occurrence, multiple representations have been used in the literature, mainly depending on the performance metrics that each approach tries to enhance. Event detection models coupled with different data structures (such as PAIS [10] in Fig. 6 that will be presented in the sequel) offer different capabilities in a CEP system. Nonetheless, all share the common goal to represent a logical or a physical plan for complex event detection in a streaming environment. In this subsection, we will review the main types of the proposed event detection models, while Table 2 compactly presents their adoption in the corresponding literature.

– **NFA.** Non-Deterministic Finite Automata is one of the most used model representations in CEP approaches (see Table 2). Each state is a partial or full detection (final state) of a complex event. The transition from one state to another is triggered when a related event, that upholds the predicates of the query, is found in the stream. An NFA example is depicted in Figures 3 and 6 where related events for state transitions are marked in the corresponding vertex (Fig. 3) or edge (Fig. 6) of the automaton. In particular, Figure 3 exhibits an enhanced variant that uses a match buffer called NFA^b which is used in [25] and [24]. The NFA model may exhibit non-determinism when at some state the formulas of two edges are not mutually exclusive, thus allowing the execution of the *skip-till-any-match* selection strategy. A slightly modified NFA, that generalizes the traditional one, is called the *Cayuga automaton* [30]. It allows the input of arbitrary relational streams with state transitions, controlled using predicates. This automaton can store data from the input stream, al-

lowing selection predicates to compare incoming events to previously encountered events.

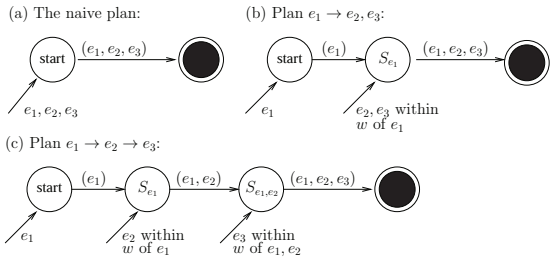

Figure 2: Finite State Machine Example (Figure from [13])

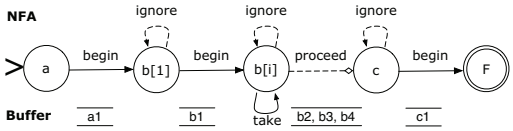

Figure 3: Non-deterministic Finite Automaton Example (Figure from [24])

– **FSM.** FSMs [12, 13] are a simpler way to express the detection of complex events. FSMs carry similar functionality with the NFAs, as transitions are made when an event that satisfies the query's predicates is detected and its states signal the partial or full detection of a complex event. In [13], FSMs are used to incorporate a plan in the detection sequence of events that, based on gathered frequency statistics, can balance latency and communication cost. An FSM example is illustrated in Figure 2 where partial matches are attached on the states of the FSM, whereas the input for each state is marked on its incoming edges.

– **Trees.** Various tree structures have been used in the literature to facilitate the detection of complex events. *Left and right deep trees* are used in [14]. A tree instance is depicted in Figure 4, where leaves are primitive events and internal nodes are operators participating in the pattern that is being examined. As we are going to discuss in the sequel, the Tree structure provides the potential for adaptivity in the query plan execution, as a transformation from a left-deep to a right-deep tree can significantly alter a query's performance based on changed occurrence frequency of primitive events.

– **Graphs.** Event detection graphs are deployed in [13, 15, 21, 22] to merge all rules for complex event detection in a single structure. Events shared by multiple expressions have single appearance in the graph. This structure offers a general overview of all existing complex expressions but is not used as a query plan for the event query execution. It rather serves as a complementary data structure that aims to reveal event dependencies.

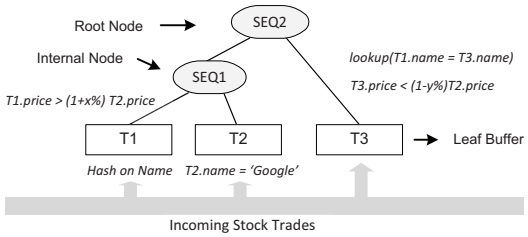

Figure 4: Left Deep Tree Example (Figure from [14])

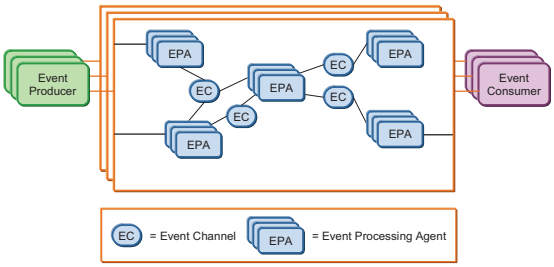

Figure 5: Event Processing Network (Figure from [23])

– **Networks.** Event Processing Network (EPN) is another way of planning query processing for detecting complex events as proposed in [11, 23]. An event processing agent (EPA) is a processing element that applies logic to incoming events and outputs derived (complex) events. The EPN follows an event driven logic where EPAs communicate asynchronously through event channels (EC). ECs are also processing elements with the responsibility of making routing decisions since they may receive multiple event streams. An illustration is given in Figure 5.

Eventually, the work in [27] introduces two variants of NFA representation, namely Chain-NFA and Tree-NFA models. The latter combines NFAs with tree-like structures, but both variants are destined to enable the potential for *lazy* query evaluation as will be discussed in Section 3.2.

2.5. Performance Metrics

Performance metrics in the context of complex event processing aim at measuring the efficiency of the query plan execution. CEP approaches that assume all data is delivered in a single source for processing, measure their performance based on how fast they process incoming data. Hence, the main optimization metric is *throughput* i.e., how many event tuples per time unit are processed. Another performance metric, highly correlated with throughput, is the *time (or CPU) cost* which is interpreted in the amount of time required for operator execution.

Techniques that assume a collection of streams arrive at different computing sites having a central (coordinator site) which dictates (based on a plan) which primitive event is monitored in which site, opt for the reduction of the *communication cost* under

Approach	Optimized For				
	Throughput	**CPU/Time Cost**	**Comm. Cost**	**Latency**	**Accuracy**
Moller et al [12]	✓				
SASE [10, 24, 25, 19]	✓	✓			
Adkere at al [13]			✓	✓	
ZStream [14]	✓				
Kolchinsky et al [27]		✓			
Wasserkrug et al [22, 15, 21]	✓				✓
Rabinovich et al [28]	✓			✓	
Lahar [16]	✓				✓
Chuanfei et al [20]	✓				✓
Shen at al [18]	✓				
Wang et al [11]	✓				
CEP2U [17, 29]		✓			✓

Table 3: Optimization Objective per Approach

the counterweight of detection latency [13]. In such cases, *detection latency* is the time between the occurrence of a complex event and its detection from the coordinator.

Another important issue upon handling large time windows is that of *memory management* (or *buffer utilization*), since, if proper care is not taken, intermediate results or partial matches stored in main memory may excessively grow.

In the uncertain context, the *Accuracy* of the inference procedure that will be made on the final outputs is an additional performance objective. Restricting the evaluated tuples based on a confidence threshold, as described in Section 2.2, results in approximate inference as the output of event queries is not complete. As a consequence, achieving appropriate balance between the conflicting goals of reduced CPU cost or data transmissions (in distributed settings) versus accuracy requirements calls for proper optimization functions to be taken into consideration.

Table 3 outlines the optimization objectives considered in the experimental evaluation of each of the techniques we examine throughout our study. An important distinction among the latency measured in [28] and the one in [13] is that, [28] measures the processing latency, while [13] the lag spent for on demand transmission among geographically dispersed sites. We stress that in any of the two cases, the CEP application may have posed restrictions on the event detection latency as a quality-of-service (QoS) requirement [31, 13].

3. Deterministic Model - Centralized Techniques

The architectural scheme of the techniques presented in this section entails a sole stream received by a single processing site, abiding by the deterministic event model.

Thus, the main performance goals involve high throughput or low CPU cost and memory consumption. These techniques attempt to cope with the Volume and Velocity aspects of the Big Data nature of event streams. In the following subsections we will make an effort to group the main optimization trends, also outlined in Table 4, that serve as the means to achieve the aforementioned objectives. Table 5 provides a concise summary of the complexities of the respective approaches.

3.1. Predicate-Related Optimizations

A simple, yet powerful, way to optimize performance is to evaluate predicates and time windows early in a query plan. This can be achieved by partitioning the input stream and by employing early filtering in the selected events that will actually be part of complex event detection based on the query. Other relevant approaches include hash-based lookups and early filters deployment on value predicates as events arrive on-the-fly.

3.1.1. Pushing Predicates Down

One of the first complete Complex Event Processing framework was SASE[10]. SASE presented a novel event language that supported all the properties of a time-ordered stream of tuples (events) and the basic operators for query specification. SASE pushes predicates down by partitioning an event stream to many small ones. Events in each partition have the same value for an attribute used in a single equivalence test within the posed query. Before we explain how it works, we should present the basics of the proposed query plan. The basic component of their query plan is sequence scan and a construction called SSC. SSC transforms a stream of events into a stream of sequences, with each sequence being a partial match of the query. For sequence scan they use Non-Deterministic Finite Automata to represent the structure of an event sequence. In sequence scan, for each partial match, an NFA is created by mapping successive event types to successive NFA states. To keep track of all simultaneous states, a runtime stack is instantiated to record the set of active states and how this set leads to a new set of active states, as an event arrives, with the use of pointers. Sequence construction is invoked when an accepting state is reached during sequence scan. They propose that sequence construction is performed by extracting from the runtime stack a single source Directed Acyclic Graph. The DAG starts at an instance of the accepting state in the rightmost cell of the stack and traverses back along the predecessor pointers, until reaching instances of the starting state.

The authors propose using an auxiliary data structure, called Partitioned Active Instance Stack (PAIS). Thus, they simultaneously create the partitions and build a series of these stacks for each, without incurring any overhead to the events that do not participate in the query. With PAIS sequence construction is only performed in stacks in the same partition, thus producing less intermediate results as shown in Figure 6.

When *multiple equivalence tests* are present in a query, the authors propose two different filtering approaches. The first, called *multi-PAIS*, performs aggressive cross-attribute (NFA) state transition filtering in sequence scan. The second approach, called *Dynamic Filtering*, pushes the most selective equivalence test down to sequence scan and then pushes all other tests to sequence construction. This approach cannot filter as

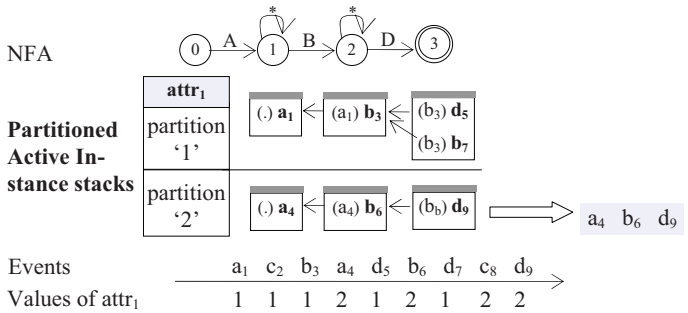

Figure 6: PAIS Example (Figure from [10])

many events in sequence scan, thus having more instances in the stacks, but does not need to pay the overhead of cross-attribute transition filtering and multi-stack maintenance.

The same principle can be applied to simple predicates and time windows. Again, there are two approaches for *pushing time windows down* which are not mutually exclusive. The first is to apply windows in sequence scan and the second in sequence creation. The former filters some of the events, so that they are not included in the stacks, and the latter searches those stacks and checks the time window on-the-fly for each event sequence.

3.1.2. Postponing with Early Filters

[24] uses an optimization to prune inconsistent events based on value predicates on the Kleene Closure operator. Although predicate evaluation is basically performed (or postponed) during edge evaluation for state transition in the NFA model that they adopt, they propose an additional filtering technique to prune repeating events, based on an existing predicate on them. They advocate that it is incorrect to evaluate all value predicates on-the-fly as events arrive, since there is the case of non-deterministic actions on the NFA model. The latter may result in multiple results when using the *skip till any match* selection strategy. Hence, they categorize the value predicates based on their consistency into 4 categories that can be applied to the Kleene Closure operator.

– **True-value consistent predicates.** Such a predicate denotes that when the result of the current event compared to all selected events is true, then it is always true and it is safe to include such an event without further evaluation of its predicate. For instance, in pattern SEQ(a, b+, c) predicate b[i].val $>$ max(b[1..i-1].val) once true remains like this for a particular $b[i].val$. Other events that do not satisfy this property are marked as unsafe, not discarded but re-evaluated in the result construction phase.

– **False-value consistent predicates.** Events that are evaluated as false under such a predicate, compared to the selected events, can be immediately discarded as they will always be false and never qualify. As an example, in pattern SEQ(a, b+, c) predicate c.val$>$max(b[1..i].val) is and remains false for the current instance of event type c. Other events must be checked again in the result construction phase.

– **True and false-value consistent predicates.** Events of this category are safe if they pass the predicate evaluation, or discarded if they do not. For instance, in pattern SEQ(a, b+, c) predicate b[i].val>5.

– **Inconsistent predicates** are predicates that are neither true-value nor false-value consistent. This category's pruning is postponed until the result construction phase. As an example, in pattern SEQ(a, b+, c) predicate b[i].val > avg(b[1..i-1].val).

Based on the above categories, it can be decided whether a predicate can be evaluated on-the-fly and prune nonconforming events, or such an evaluation must be postponed until the result creation phase.

3.1.3. Hashing for Equality Predicates

[14] addresses the issue of equality predicates and multi-class equality predicates. They propose the use of hash-based lookups, whenever it is possible, in an effort to reduce search costs for equality predicates between different event classes (types). Multi-class predicates can be attached to the operators as other predicates.

The mechanism employed involves the use of buffers of events in all nodes of the utilized tree model representation, with the leaves being primitive events and the internal nodes being operators. Based on the equality predicate, a hash table is created in the buffer (of the leaf node that takes part in the equality predicate) as events arrive and when the predicate is employed (further up in the tree) hash lookups are performed to prune results, that do not conform with the predicate.

Hash tables are created on the buffers of leaf nodes, either to some or to all the involving events depending on the used operator. If the operator is AND, then a hash-table is created on all participating event buffers. If the operator is SEQ a hash table is created to the first event in the sequence that takes part in the predicate. Hash construction can be easily extended with *multiple equality predicates*, where the use of a secondary hash table is possible for sequential patterns to facilitate faster pruning of irrelevant events.

3.2. Query Rewriting/Reordering

Query rewriting is a popular optimization technique that allows a non-optimized query expression to be rewritten in a more efficient one. The rewritten query must produce exactly the same results as the original one and must exhibit enhanced performance upon the optimization objectives. There are several papers in the literature that adopt this optimization technique and though [12] poses as a distributed solution, since optimizations are performed in a central node, here we view it as a centralized technique.

[12] addresses the issue of query rewriting in three operators $Union$, $Next$ and $Exception$, that have the same functionality as the aforementioned OR, SEQ and NOT respectively. The basic intuition is to transform the original patterns to equivalent ones with lower CPU cost. For the $Union$ (OR) operator, since the detection of the complex event relies on detecting one involved event without any order, different ordering of the events may result in more efficient evaluation. Based on the commutative and associative property of this operator, the authors map the problem to finding the optimal

prefix code for compression of data, i.e., a set of bit strings in which no string is the prefix of the other. A prefix code is also represented by a binary tree, where each internal node is a one or a zero and each leaf represents the frequency of the character. They use the greedy *Huffman* algorithm to create that tree and the optimal order of the $Union$ patterns is generated with a depth first traversal. The $Next$ (SEQ) operator also has the associative property but not the commutative one, i.e., the order of the events cannot be altered. The authors propose a dynamic programming solution where the lowest cost pattern can be found by enumerating all equivalent patterns and computing each cost. The same principle applies for the $Exception$ (NOT) operator and therefore the same algorithm with the $Next$ operator is used.

[14] tackles the issue of query rewriting using algebraic rule-based transformations. The transformations, performed by the authors, differ from the solution proposed by [12] since the basic transformation is algebraic and thus can alter the used operators. As an example, the expression SEQ(A, AND(!B, !C), D) is equivalent with the expression SEQ(A, !OR(B, C), D) where ! is the NOT operator. They implement a series of such equivalence rules that can generate an exponential number of equivalent expressions for a given query pattern. Instead of searching all those expressions exhaustively, they transform the query when the rewritten expression has a smaller number of operators or the expression contains lower cost operators. The higher cost operator is the AND operator, followed by the SEQ operator and the less expensive one is the OR operator based on their cost model. After the query is simplified the authors propose an algorithm for reordering the operators in a similar conceptual way as in [12]. Using a dynamic programming algorithm, they try to find the optimal order of the events, that are included in the query. The optimal order though in this implementation can be found in a set of different (left-deep, right-deep, bushy) tree structures, which are derived from a single logical query plan. They use an algorithm which essentially enumerates the power set of the operators.

The technique proposed in [27] does not explicitly rewrite a given event query, but silently reorders the incorporation of incoming events in partial matches until the right moment comes. More precisely, the novel lazy evaluation approach proposed in [27] utilizes incoming events on partial or full pattern match evaluation not according to the order by which event type instances arrive, but in increasing order of event type selectivity. That is, events of more frequent event type participating in a posed query are buffered until the rarer event type instantiations are received. After the reception of the rarer event, the evaluation of the infrequent and buffered events is performed to determine potential pattern matches. In order to perform the lazy evaluation approach two NFA variants are invented. The first one, termed Chain-NFA, is used in cases where selectivity of events is known in hand. The second, termed Tree-NFA, uses a tree like structure to dynamically and adaptively route incoming events according to their frequency (selectivity). Interestingly, Tree-NFAs are useful when selectivity is not known in hand and may be learned and altered on-the-fly as new events arrive.

[28] presents an assertion-based pattern rewriting framework for two operators, namely all (AND) and $sequence$ (SEQ). They advocate the splitting of patters into disconnected components that can be independently processed. The acquisition of the disconnected components is achieved through the following steps: (a) The pattern is converted into conjunctive normal form (CNF) by employing De Morgan laws, (b) a

variable dependency graph is created, to recognize independent components, where variables in the WHERE clause of the query are represented by nodes and their connection by edges, (c) the pattern is split into maximal number of independent partitions which imply the finest granulation that can be performed. This work combines both rewriting (as in [14]) and reordering (as in [12]) to enable query granularization.

It worths noting that the approaches of [14, 27] are the only ones that include provisions for adapting the produced query plans to changing data distributions, as marked in Table 4. In order to recompute the plan on-the-fly, [14] maintains a running estimate of event statistics, using sampling operators attached to the leaf buffers of the utilized tree structure. In this implementation, simple windowed averages are used to maintain the rates of each input stream and the selectivity of each predicate. When any statistic used in a plan varies by more than some error threshold, the operator ordering algorithm is rerun. A new plan is installed, provided that the performance improvement predicted by the cost model is greater than another, performance related, threshold. On the other hand, the Tree-NFA representation of [27] aids in dynamically routing events on-the-fly, according to their selectivity.

3.3. Memory Management

Since the input to CEP approaches is an infinite stream of events and queries are bound with time constraints based on the specifications of the WITHIN clause, intermediate results (partial matches) can grow exponentially. They can, therefore, fill all available memory since time windows can be as large as desired. The memory management feature though is not handled explicitly in most approaches, since it also affects system's throughput in most cases. We choose not to categorize as memory management optimizers approaches inventing efficient indexing structures [10, 14] in their effort to optimize throughput or other criteria, since these approaches do not directly aim at shedding the storage load.

On the contrary, the shared buffer approach presented in [25] does constitute a mechanism aiming at reducing memory usage. [25] uses buffers to encode partial and complete matches for each query run. The basic principle is to share both storage and processing across multiple runs in the NFA-based query plans. The initial approach is to build a buffer for each single run and then merge such individual buffers into a shared one, for all the runs.

Each individual buffer contains a series of stacks, one for each state of the NFA except the final state. Each stack contains pointers to events that triggered a transition in the NFA's state and thus are included into the buffer. Further, each event has a predecessor pointer to the previously selected event in either the same stack or the previous one. For any event that triggered a transition to the final state, a traversal across the predecessor pointers reveals the full detection of a complex event, as specified by the evaluated query.

Consequently, those buffers are combined into a single shared one to reduce the memory and processing overhead. This process is based on merging the corresponding stacks of individual buffers, by merging the same events in those stacks while maintaining the predecessor pointers. This process though can result in erroneous results, since it is unable to distinguish predecessor pointers from different runs. To alleviate this problem an identifier number(version) for each individual run is used to label all

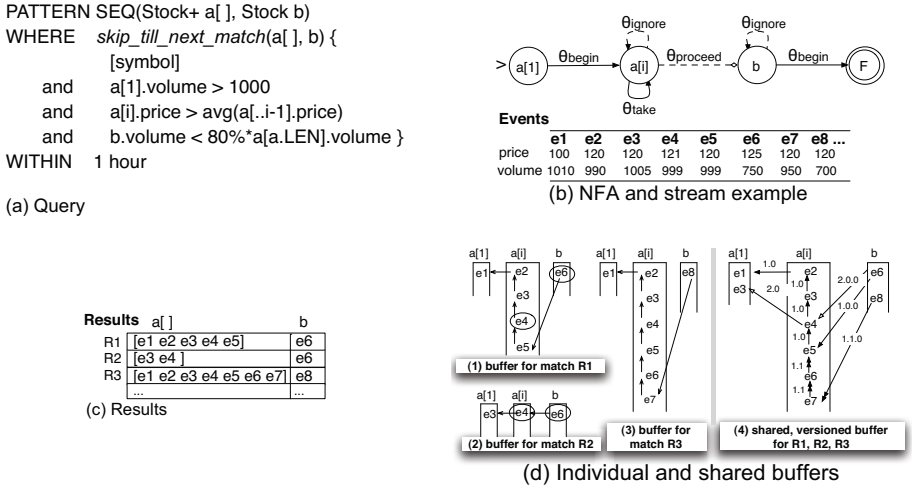

Figure 7: Query evaluation and shared buffer creation example from [25]

such pointers created in that particular run. An additional issue is that runs cannot pre-assign version numbers, since non-deterministic states can spawn new runs at any time. Thus the version number is dynamically grown as the run proceeds.

This technique guarantees that the version number is compatible with a possibly ancestor run (spawned by it), since they would share a common prefix. The versioned shared buffer can thus encode compactly all possible runs. To extract a detection of a complex event, the algorithm takes the version number of the run and traverses from the most recent event (in the last stack) along the compatible predecessor pointers to the event that started the run. A detailed example with a query (a), its respective NFA and an event stream example (b), the generated results (c) and the generation of the versioned shared buffer from the individual ones (d) is illustrated in Figure 7.

We conclude this section with Table 4 which compactly presents the main optimization strategies adopted by each of the approaches discussed so far, as well as those that will be presented in the following section. It can be easily observed that predicate-based pruning and query rewriting/reordering are the most well studied optimization means. As Table 4 shows, efficient buffer usage is explicitly considered only in SASE, whereas the rest of the optimization strategies cited in the table are dedicated to serving certain approaches.

4. Probabilistic Model - Centralized Techniques

The current section reviews approaches that have been proposed in the literature of CEP under uncertainty. In that, apart from the Volume and Velocity aspects, we also concentrate on the dimension of Veracity in Big Event Data. As in Section 3, we classify corresponding techniques to centralized due to the fostered architectural scheme of the underlying network that includes a single processing node. Surveys especially focused on uncertainty issues can also be found in [7, 8]. Here, we concentrate on aspects closely related to query processing and optimization in the probabilistic context.

Again, Table 4 outlines the optimization strategies adopted by the approaches discussed in this (and the previous) section, while Table 5 summarizes their complexities. Before presenting specific approaches, we begin our study by discussing the impact of uncertainty on query processing compared to what has already been presented.

4.1. Impact of Uncertainty on Query Processing

Consider the case where uncertainty in event occurrence, as defined in Section 2.1.2, is examined. Content uncertainty manipulation is analogous upon having a mechanism, as the one described in [17], by which uncertainty in event content is interpreted to uncertain event occurrence for composite events. In Section 2.1.2 , under uncertain event occurrence, we made the convention that any given event is represented along with its occurrence probability $< e, p_e >$. The complementary case, that of e not occurring, can be denoted as $< \neg e, 1-p_e >$. Now, assume that the pattern operator, against which complex events are detected, entails n event types in its input list (also see Section 2.2). In general, an event tuple may possess more than two probabilistic instances [19], but even for two distinct cases for a single event occurrence there exist up to 2^n possible instances of inputs for the chosen operator that need to be examined for matching the given pattern. Each such instance is considered a *possible world*. As a consequence, the actual input to the evaluation process of an event query is a set W of cardinality $|W| = 2^n$, instead of a single event instance in the case of deterministic event model. Previous works [20, 16, 18, 19, 11, 17] employ the model of possible worlds in order to perform event query evaluation.

Having elaborated on the way uncertainty affects the input of query evaluation mechanisms, we then focus on the internals of query processing. Techniques on pruning intermediate results during pattern matching evaluation, such as those presented in Section 3, are still applicable in order to exclude possible worlds whose instances are inconsistent with the examined query pattern. These techniques, though, defy the uncertainty dimension. A form of a predicate that is tailored for probabilistic event handling is that of a confidence threshold that can be incorporated in the posed query to filter possible worlds that are highly improbable, as discussed in Section 2.2.

A direct effect of that HAVING clause is that intermediate stages of pattern evaluation can output confidence values below the given threshold thus enabling the CEP system to prune further evaluation of the corresponding possible world. The ability of pruning intermediate results based on the above confidence parameter is explicitly taken into consideration in SASE+ inspired techniques in [19, 18, 20, 11] and is noted in Table 4 as a predicate-based pruning (optimization) technique. The confidence value of a pattern operator expresses the occurrence probability of a complex event in the RETURN clause of the above query, which in turn outputs a number of possible worlds for the complex event it detects.

Admitting uncertainty in primitive events requires the definition of a mechanism that maps the probability values attached on them to uncertainty quantifications on the derived, complex events. Taking one step further, a similar formula needs to be utilized to estimate the uncertainty of complex events that are comprised of other complex events and/or primitive ones. Given an operator such as SEQ, AND, OR and so on, a matched pattern responsible for the detection of a complex event receives

propagated uncertainty values based on the assumptions that are made regarding the (in)dependence of the events it receives as input.

Different uncertainty propagation assumptions may affect memory usage. We thus review the assumptions existing uncertain CEP techniques employ to attach probability values to complex events. Due to its simplicity the most popular assumption employed in [21, 15, 22, 19] is the *event independence assumption*, at least up to the level of primitive events. [17] considers both primitive and complex events to be independent since it considers general event hierarchies and traces, where existing dependencies would severely impact the complexity of processing. The approaches in [16, 11] employ the *Markovian hypothesis* to propagate uncertainty among events. That is, the probability of an event occurrence depends only on the probability of the very previous event (given an event selection strategy) and not on the sequence of events that preceded it. Eventually, the works in [17, 21, 15, 22] foster *Bayesian Networks (BNs)* to propagate uncertainty along the rules describing complex events, while [20] uses Bayesian formulas to determine the uncertainty of complex events. In a nutshell, a BN is a Directed Acyclic Graph (DAG) in which nodes represent random variables and an edge between nodes represents a dependency between them, in the sense that the event at the beginning of the edge affects the event towards the direction of the edge.

4.2. Frameworks Considering Ubiquitous Uncertainty

We first elaborate on techniques that account for all uncertainty flavors discussed in Section 2.1.2, from content uncertainty, to uncertain event occurrences and uncertain rules. This category includes the works of [17, 21, 15, 22]. The framework that was initially modeled in [21] and later enhanced in [15, 22] assumes independent primitive events, while dependencies among primitive and complex as well as between complex events are inferred based on BNs. A basic structure that is employed involves a triggering graph as marked in Table 2. This graph expresses the rules (corresponding to operator matches in the previously discussed frameworks) under which complex events, either derived from primitive ones or lying in nodes of complex event hierarchies, are triggered (occur). The triggering graph together with a quasi-topological order, defines the triggering order or stated differently the required order of events' occurrence. Such a graph is static and represents event types and their relationships that form the rules. An algorithm is proposed which, as events arrive in the CEP system, automatically and progressively constructs a Bayesian network examining a particular rule's instantiation. BNs essentially instantiate parts of or complete rules expressed by the triggering graph by considering event instances instead of generic event types. Furthermore, they quantify event dependencies in Conditional Probability Tables (CPTs). However, the CPTs and the BN may be constructed from scratch with each new event, due to the non-monotonic reasoning of the rules, which is inefficient. To improve efficiency, a sampling algorithm is proposed to obtain a sample from the BN without actually constructing it. Instead, a sample for the primitive events is generated using the mutual independence assumption. Then, the algorithm traverses the rules in the triggering graph and triggers events according to each rule, given the sampled event history. In addition, the instantiation of complex events according to each rule is based on probabilistic rule definitions. The proposed sampling algorithm obviously introduces an approximate inference process to improve efficiency (throughput in Table 2)

on the counterweight of sacrificing accuracy. However, this kind of approximation differs from the one caused by the confidence threshold in the HAVING clause of the posed query (Section 2.2 & Section 2.5). As already discussed, the latter constitutes a predicate-based pruning scheme, while the sampling algorithm of [15] heuristically assesses materialized event probabilities. Hence, it is noted as a separate optimization means in Table 4.

The CEP2U model discussed in [17] also considers content, occurrence as well as rule uncertainty. For each uncertain attribute of an event, the information received by the CEP engine is a couple: $< value, pdf >$ where $value$ is the observed value of the attribute and pdf is its probability density function. Primitive events hold zero uncertainty about their occurrence, but possess uncertain attributes, while the probability of occurrence of complex events is derived from the uncertainty of primitive events' attributes. The uncertainty in attributes of complex events is again derived from their ancestors by combining their pdfs under the independence assumption. Eventually, uncertainty in rules is modeled, similarly to [15], utilizing BNs which are constructed independently of the previously mentioned uncertainty propagation process. In other words, CEP2U produces two probabilities of occurrence for complex events and hierarchies. One by consulting uncertainty related with events and one by independently examining the uncertainty coming from rules modeled by BNs. The two are synthesized by computing their product. The result is the uncertainty of the complex event occurrence. CEP2U's model is validated by a corresponding implementation in the T-Rex engine [29]. To match incoming events based on posed predicates an algorithm is used that evaluates constraints and their satisfaction probability sequentially against each incoming event. Additionally, using the functionality of T-Rex [29], CEP2U can make use of a thread pool to process multiple rules (patterns) in parallel.

4.3. Approaches Considering Temporal Uncertainty

The SASE-inspired framework of [19] focuses on the uncertainty embodied in the temporal reference of streaming events. A time interval is used to bound the occurrence time of each event and the timestamps of different events are assumed to be independent of each other. The set W of all possible worlds is derived by breaking, based on a given time granularity, events' time intervals to distinct time values of so called point events. The authors initially present their point-based framework that entails three steps. A stream expansion step by iterating all over the possible timepoints in events' intervals. A pattern matching step follows where, given a point event that generates an initial partial match of a pattern, "skip till any match" (see Section 2.3) is used to dynamically construct a DAG. The latter DAG is rooted at the given event and spans the point event stream, such that each path in this DAG corresponds to a unique match starting from the root. A match collapsing phase first collects matches as they are produced by the pattern matching module and groups them based on their signature (unique sequence of event ids in a match). It then collapses them to a particular format before confidence values can be calculated. Since this basic approach results in enumerating exponentially many point based matches, a second event-based evaluation framework is proposed. In the latter framework, the algorithm conceptually walks through the DAG of events three times: first forward according to the pattern, examining the tightness of the lower bounds of event intervals belonging to a potential match; second backwards,

Approach	Optimization Strategy						
	Query Rewriting/ Reordering	**Predicate Related Optimizations**	**Memory Mgmt**	**Coordinated Push/Pull**	**Parallelization**	**Approximation**	**Comments**
Moller et al [12]	✓				✓		Operator Distribution
SASE [10, 24, 25, 19]		✓	✓				Temporal Uncertainty Handling [19]
Adkere at al [13]	✓	✓		✓			Distributed Network Setting
ZStream [14]	✓	✓					Adaptive
Kolchinsky et al [27]	✓						Lazy Evaluation, Adaptive
Wasserkrug et al [22, 15, 21]						✓	Uncertainty Handling, Sampling BNs
Rabinovich et al [28]	✓						
Lahar [16]	✓						Uncertainty Handling
Chuanfei et al [20]		✓				✓	Filter with Tolerance, Uncertainty Handling
Shen at al [18]		✓					Uncertainty Handling
Wang et al [11]		✓			✓		Distributed Network Setting, Uncertainty Handling
CEP2U [17, 29]		✓			✓		Uncertainty Handling

Table 4: Optimization Strategies Adopted by Each Approach

for doing the same with the upper endpoints of each event interval, and third backwards again, checking the consistency of the matches with the time-based query window. A third framework is introduced considering events in their arrival, instead of query, order.

In the same spirit, the recent work of [32] loosens the event independence assumption made in [19], considering dependencies among some of them. [32] proposes an index structure for event data where the temporal dependency relationship of events is captured using a set of integer codes. These also aid in deducing the ordering of events within formed dependency groups. An additional structure is used to index all the attributes and time intervals of the events. Based on the above pair of indexes, algorithms are developed to efficiently extract patterns from the event data.

4.4. Query Rewriting in the Probabilistic Context

The Lahar CEP system proposed in [16] presents an approach for evaluating pattern queries over correlated streams. In Lahar's approach primitive events streamed into the system are tagged with their corresponding probabilities. Two modes of operation are considered: real-time or archived. In the real-time scenario, the system does not possess correlation information among streams. Thus, it assumes stream independence, but also Markovian correlations for events belonging to the same stream. On the other

hand, in the archived scenario Markovian streams are considered. A query of interest is broken down to a number of subgoals and query classes are identified based on the variables that are shared amongst the subgoals. Efficient algorithms for processing each query class are accordingly proposed. The first class of queries involves those termed as *Regular queries*. Regular queries mainly consist of selections and do not share any variables among their subgoals. They are translated to regular expressions using a four step procedure which initially defines a set of symbols on which a simple automaton operates. Then, the W set of all possible worlds is translated into a sequence of subsets of the previous symbols, while a third step involves query translation into a regular expression. Finally, the distribution of the Markov chain that is induced by W is recovered and used to evaluate the regular expression. Regular query evaluation can be made in real time since Regular queries' evaluation cost is linear to the number of possible worlds. Moreover, their evaluation can be made in an incremental fashion. *Extended Regular queries*, apart from selections, allow for joins and projections and possess shared variables across all their subgoals. As a result, they can be decomposed to regular queries and be processed independently, synthesizing probabilities in the final outcome based on the independence assumption. *Safe queries* involve more complicated projections and sequences, while *Unsafe queries* are hard to evaluate and thus an approximation algorithm is proposed. Both Safe and Unsafe queries are only destined for the archived mode which goes beyond the focus of our discussion.

4.5. Event Processing with Lineage

The work in [18] considers uncertain event occurrence. Following the SASE+ query language paradigm, and more precisely concentrating on the SEQ operator, the query under study uses the (probabilistic) confidence predicate along with a corresponding threshold in the HAVING clause for qualifying simple or complex events based on the uncertainty values computed for them. Consequently, as already discussed (Section 2.2), approximate inference is performed using the predefined threshold for pruning. The work considers the possible worlds model to feed corresponding instances to a SEQ operator for testing matches. In order to perform the pattern matching, NFAs are used to represent the structure of an event sequence. To keep track of simultaneous state instances, an auxiliary data structure, namely Active Instance Graph (AIG), is employed. The AIG is a DAG connecting events with previous candidate events, i.e. whose possible occurrence may lead to the recognition of a complex event. In particular, for matching patterns, sequence scan is called when new events arrive and a search regarding the possible predecessors in the active instances is initiated. By backward-traversing the AIG, the sequences that, even partially, satisfy a complex event definition may be retrieved. Dynamic filtering over predecessors also using the confidence threshold or other predicates is performed and eventually the DAG is maintained by adding active instances that satisfy the conditions. When an accepting state is reached, sequence construction is called to generate resulted sequences. As regards the dependence hypothesis, lineage, representing sufficient dependency information of query results, captures where an event came from. Lineage is represented as a function that associates each possible world with a boolean formula whose symbols are other possible worlds involving primitive events or event sequences.

4.6. A Single-pass, Pruning and Filtering Algorithm

[20] handles uncertain event occurrences by proposing an algorithm that requires a single scan over the probabilistic stream in order to detect complex events satisfying a given event query's requirements. The work considers probabilities of sequences of events, i.e., the SEQ operator, to be computed based on the Bayesian formula. To efficiently update and query the corresponding conditional probabilities an indexing structure, namely the Conditional Probability Indexing Tree (CPI-Tree), is used. The algorithm employs NFAs to perform pattern matching and Chain Instance Queues (CIQs) to organize streaming events based on their type and order of arrival. CIQs are composed of nodes, one for each state of the respective NFA, where relevant event instances are stored. In that, CIQs' nodes resemble the stacks used in [25]. The main difference is that the links connecting CIQs' nodes head only forward with respect to the position of the NFA state a node corresponds to.

The pruning and filtering algorithm operates in two phases. During the *enter* phase, event instances that remain capable of aggregating sequences that satisfy a given confidence threshold are not pruned and are admitted to the corresponding CIQs' nodes. In the *search* phase, CIQs' nodes are scanned starting from the one corresponding to the first state of the NFA and heading forward. The algorithm builds sequences by connecting event instances of a type involved in the evaluated pattern to relevant event instances of the next event type in the pattern, meanwhile calculating the probability of each chain. A confidence threshold is used to filter out chains that are highly improbable. Eventually, an additional layer of approximate inference regards the utilization of a tolerance parameter. Tolerance is computed based on the value of the maximum conditional probability so far and the user specified confidence threshold. It is subsequently attached to the confidence threshold so as to increase the potential for pruning while ensuring high recall in the detected sequences.

5. The Distributed Case

There are two ways of distributing complex event processing. The first is to centralize the stream monitoring and distribute, or more precisely parallelize, the complex event processing to multiple sites, as proposed by [12]. The second is to distribute the stream monitoring to multiple sites (which receive multiple input streams, one per site) and centralize the processing to a central site, as proposed by [13]. Notably, the work in [11] examines both approaches in the probabilistic context, although communication related performance reports of the distributed monitoring rationale are absent from the empirical evaluation. This is denoted in Table 4 by the missing mark in the communication cost column of [11].

5.1. Distributed Processing

The work in [12] discusses the idea that distributing the query processing across multiple nodes, i.e., of a computer cluster, drastically enhances the system's performance in terms of throughput. While not considering the sources to be part of the system, communication is performed with event messages through high-rate dedicated lines. As such, the system architecture differs from the distributed monitoring of event

generating sources, that we will review in the next subsection. Although many sources can issue events, these are routed through the central node to the processing nodes. The central node has all the available information, concerning the deployed operators, but does not perform any processing of the incoming events.

The basic insight is that parallelizing the processing via distributing query operators (see Table 4) elevates the problem of memory management, as well as system throughput. Memory management is enhanced, since the partial matches (automaton instances) are distributed through many nodes, and are therefore able to deal with larger time windows. Throughput is also optimized, through the fact that each node receives less events for less queries, thus overall the system processes more events per time unit.

The main optimization in [12] is that the queries are rewritten to more efficient ones, as described in Section 3.2. The reason for grouping their optimization technique as centralized is that all the optimizations are performed in the central node and only the processing is distributed. This is performed with the use of a greedy algorithm for choosing operator deployment plans. The algorithm reuses already deployed operators and deploys the remaining operators in a bottom-up fashion. Existing operators are stored in a hash map for fast retrieval. First, a submitted query is traversed top-down to find the largest equivalent deployed operator in the hash map, if any, starting with the entire query. If found, the expression is replaced with a marker containing the operator identifier and location to allow operators to be connected once deployed. Next, the remaining operators are deployed bottom-up. The assignment of each operator is selected by recursively placing the left and right sub-expressions of the operator and then the operator itself. An operator is assigned by calculating the cost of placing the operator on each node and selecting the lowest cost node. This approach practically selects good deployment plans, but they are not necessarily optimal.

5.2. Distributed Monitoring

The work in [13] considers an architectural scheme of multiple streaming sources and a single base-processing node (see Figure 8). Each source is a receiver of an input stream of events and the base node is a coordinator node that communicates with all the sources, for detecting complex events. In such a scheme, as stated in previous sections, the main concern is that the *latency* for detecting the complex events is user (or system) specified and that the *communication cost* for communicating with the base node (coordinator) is controlled.

With the above in mind, plans are generated trying to balance the communication cost with the detection latency. The approach uses FSMs for the physical event detection plans and event detection graphs for the logical model representation of all the existing queries. The event detection graph depicts (in a single graph) all available queries, with leaf nodes being primitive events and internal nodes being operators or complex events. Using the event detection graph, *pareto* optimal plans are created that take into account event sharing across multiple queries, in the sense of *Multi-Query Optimization*, as well as event frequencies and acceptable latency values. With these optimal plans being deployed, based on the cost-latency model that they propose, they are able to generate monitoring plans (FSMs), which conform with the chosen cost and latency constraints. These plans are then distributed across the sources and through push and pull messages the coordinator detects complex events. The activation of the

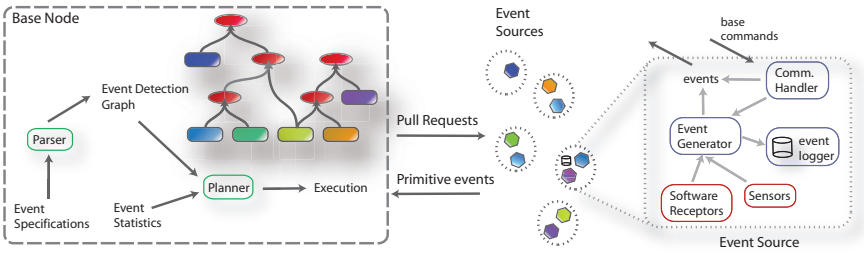

Figure 8: Distributed System Architecture (Figure from [13])

FSM's final state, after the occurrence of all the related events in the way that the query specifies, signals the detection of a complex event.

Based on the number of states of the FSM (see Figure 2), they can monitor any number of events that participate in the query. The activation of a new state (through the detection of the current state's primitive event) marks the monitoring of a new set of events. The basic idea behind the number of selected states of an FSM (which is the monitoring order of events) is that; *"processing the higher frequency events conditional upon the occurrence of lower frequency ones eliminates the need to communicate the former in many cases, thus has the potential to reduce communication cost in exchange of increased detection latency"* [13]. In that, a trade-off between latency and communication exists, by reordering or rewriting (merging) FSM states.

The plan generation is done by traversing the event detection graph in depth-first manner, running the plan generation algorithm on each node, in order to create a set of plans with a variety of cost and latency characteristics. At the parent node of each complex event, where all plans are propagated, the selection of the plan marks the selection on the children nodes. This hierarchical plan composition takes also into account shared primitive events by multiple queries, so that it will be taken into consideration at the plan selection. The plan generation algorithm is a Dynamic Programming algorithm that achieves the minimum global cost for a given latency value, but has exponential time complexity and thus is applicable to small problem instances. This is the reason that the authors also applied a heuristic algorithm that runs in polynomial time and, although it cannot guarantee optimality, practically produces near optimal results.

Plan execution then commences by activating all the FSM's starting state that triggers the continuous monitoring of some primitive events at the various sources, by informing them which events are of interest by the coordinator. Once such an event is detected at a source node, it is *pushed* to the coordinator who makes the transition in the respective FSM to the next state and then *pulls* from the source nodes the next primitive event that the next state monitors. Once a final state is reached, inside the time window specified by the query, the coordinator detects a complex event.

Another optimization employed in this paper is the use of spatial and attribute-based constraints. The main intuition is similar to the centralized optimization that "pushes predicates down" (Section 3.1), but this time the efficiency objective regards reduced communication. By evaluating the predicates before the events are sent to

the central node, unqualified events are pruned and do not add overhead to the overall system performance.

5.3. Distributed Monitoring over Probabilistic Event Streams

The sole technique considering probabilistic CEP monitoring over a number of distributed sources appears in [11]. In particular, [11] discusses both distributed processing in terms of parallelization, as [12] does, but also distributed monitoring in a fashion similar to [13]. The work proposes algorithms for parallel, distributed and hierarchical complex event detection under uncertainty. In particular, it takes into consideration uncertainty in *event occurrence*, supporting approximate inference by the definition of the desired confidence threshold in the HAVING clause of the posed query. Primitive event instances from different streams are considered independent; however, events of a single stream evolved on a SEQ operation are assumed to possess the Markov property.

In its algorithmic part, [11] extends the basic building blocks of [18], that is Non-Deterministic Finite Automata and the Active Instance Graph (AIG) structure. Nodes in an AIG graph correspond to Active Instance Stacks (AISs) of events that are received by the CEP system. As they arrive, events of the same type are stacked in an AIS, which is a node in the AIG, based on their event type and the order of arrival.

Initially, to enable parallelization, a plausible observation is made that the events which are related to state j of the NFA in the AIS of a partition P_i can be linked to the events which are related to state $j-1$ in the AIS of partition P_{i-1}. That is because the timestamp of the events in P_i is larger than the timestamp of the events in P_{i-1}, under the assumption that input events are ordered and partitions are based on timestamps. In order to be suited in the context of uncertain event processing, the links connecting AIS nodes are tagged by the corresponding probability values computed based on the Markov property. The new structure is called Probabilistic AIS (PAIS) and resembles the partitioned structure of Figure 6 with probability values attached in each stack element.

As regards the complex event detection over distributed streams, using the same structures, the moving temporal result approach is dictated as the preferable manner to accomplish the task. According to this approach, each node separately creates its own PAIS according to the local event stream. Then, to reduce communication costs in the distributed setting, the node with the largest PAIS is selected as the main node (similar to a coordinator in [13]) and the rest of the distributed sources send their PAIS to that node. Hierarchical probabilistic CEP is handled by different pattern matching components (agents) which can be connected through channels to form an EPN.

We conclude this section by summarizing in Table 5 the complexity of the approaches presented in Section 3, Section 4 and the current section, as well. For presentation uniformity reasons, though different techniques utilize different optimization strategies (see Table 4), we abstract the algorithmic parameters using n to summarize the number of event types, automaton states or query pattern operators[2] involved in

[2]Each event type in this case can be viewed as an identity operator returning for each input event instance, the instance itself.

Approach	Time Complexity	Remarks
Devoted to Plan Generation		
[12]	$O(n\ell ogn)$ $\Theta(n^3)$ $O(nN)$	Union operator - Greedy Algorithm Next operator-Dynamic Programming Algorithm Operator Distribution- Greedy Algorithm
[14]	$O(n^3)$	For optimal operator ordering
[27]	$O(2^n)$	For Tree-NFA - Dynamic instantiation advised
[28]	$\underline{O(n^3)}$	Offline reordering cost by enumerating the power set of operators [14] , co NP-complete for all potential rewritings apart from reorderings [33]
[13]	$O(2^{2n}nk)$	Dynamic Programming Algorithm - Greedy advised k stands for the maximum number of optimal plans
Devoted to Query Evaluation		
[10][18][17, 29]	$O(2nS)$	Sequence Construction Algorithm
[16]	$O(S)$	Regular & Extended Regular Queries
[25]	$O(2^S S^{n-k})$ $O(S^{k+1})$	In the presence/absence of "skip till any match" k stands for the number of Kleene plus states
[19]	$O(S^n)$	Pattern agnostic - For skip till any/next match
[20]	$\underline{O(nS)}$	Filtering, IPF-DA Algorithms
[11]	$\underline{O(NnS)}$	Link Creation Algorithm
[15]	$\underline{O(nm^S)}$	Abstracted for BN construction - Sampling advised (see[15]) m :size of the state space of each event

Table 5: Complexity per Approach. n: number of event types, automaton states or query pattern operators, S: the input size, N: the number of processing nodes or sites. Underbars mark estimated complexities.

a query. S denotes the input size which is a (potentially unbounded) window of observations coupled with the number of possible worlds for probabilistic techniques. Obviously, $n \ll S$ holds and in practice n receives a small value even for more complex queries. Furthermore, we use N to denote the set of processing nodes or sites for distributed approaches. Eventually, we make an attempt to extract the complexity of approaches that do not explicitly report it by studying their algorithmic description. We stress, however, that this is just our estimation which we mark by underlining respective references in Table 5. The work in [24] presents an extensive study on performance bottlenecks, analyzing the complexity of event pattern queries and commenting on features that render event queries computationally expensive.

6. Issues on CEP in Cloud Computing Platforms

In our study so far we dealt with techniques opting for efficient query processing over unbounded, high speed event streams. Moreover, we integrated studies on uncertain events as well as distributed architectural settings with geographically dispersed processing nodes. In that, we described the status and capabilities of current approaches regarding the manipulation of the Volume, Velocity and Veracity aspects of Big Event Data as promised in the introductory part of our work. Nonetheless, Big Data also requires exceptional hardware technologies to efficiently process large quantities

of data within tolerable elapsed times, the characteristics of which are not explicitly taken into account by the techniques discussed so far.

The cloud computing paradigm entails access to such hardware infrastructures, more directly via the Infrastructure-As-a-Service (Iaas) capability, that serves as a pool of configurable computing resources. These resources include, but are not limited to, enhanced processing power in multiple nodes, high memory capacity and high speed network connectivity. With the emergence of streaming cloud platforms (such as S4 [3], Storm [4], Spark [5] and so on) the way efficient event query processing in the cloud computing paradigm is going to be achieved needs to be rethought. In order to take the maximum benefit from the cloud, thus optimizing query processing, the basic properties any candidate CEP technique should possess are closely related to: (a) parallelism, in order to take full advantage of the availability of multiple computing nodes, and (b) the potential for *elasticity*. In a nutshell, elasticity refers to the fundamental property of dynamically adding or removing resources devoted to the execution of a query.

First, with respect to the first of the above characteristics, it is easy to conceive that parallel algorithms for efficient query processing are capable of increasing throughput by taking advantage of multiple processing nodes. Due to this fact the performance objective, i.e. throughput or time cost is not altered compared to what was discussed in Section 2.5 and summarized in Table 3. An additional criterion however relates to the *utilization* of the processing nodes provided by the cloud platform. Overly high utilization leads to low throughput which may in turn cause violations of time requirements (restrictions) for resource occupation [31]. On the other hand, under-utilization leads to inefficient resource usage increasing *monetary costs*[6].

Second, the notion of elasticity resembles the adaptivity concept discussed in Section 3.2 for the approaches in [14] and [27], in the sense that it adjusts the query plan by adding/removing resources [14] or by shuffling [27] event tuples based on selectivity estimates. However, beyond that, elasticity distinguishes cloud computing from other paradigms, such as cluster or grid computing, by also aiming at matching the amount of resources allocated to the (event) processing task with the amount of resources it demands, avoiding over- or under-utilization. The degree to which the aforementioned matching is achieved may be measured in various metrics [34].

Additional features desired in a cloud setting involve the need for *multi-query* optimization approaches over cloud platforms. This becomes a necessity so as to provide query plans that are optimal for the whole set of event processing tasks of a particular end user, thus alleviating monetary costs. To complete the picture, the realization of CEP over planetary-scale Big Data calls for distributed, i.e. including a number of dispersed resource pools, streaming cloud architectures [35].

Given our previous discussion, Table 6 summarizes what we choose to term "cloud-friendly characteristics" of the CEP approaches discussed in the previous sections. As the table demonstrates, the vast majority of the discussed techniques totally ne-

[3] http://incubator.apache.org/s4/

[4] http://hortonworks.com/hadoop/storm/

[5] https://spark.apache.org/streaming/

[6] For example, see https://cloud.google.com/pricing/

Approach	Parallelism	Adaptivity	Distributed Network	Multi-query Optimization
Moller et al [12]	✓	×	×	×
SASE [10, 24, 25, 19]	×	×	×	×
Adkere at al [13]	×	×	✓	✓
Zstream [14]	×	✓	×	×
Kolchinsky et al [27]	×	✓	×	×
Wasserkrug et al [22, 15, 21]	×	×	×	×
Rabinovich et al [28]	×	×	×	×
Lahar [16]	×	×	×	×
Chuanfei et al [20]	×	×	×	×
Shen at al [18]	×	×	×	×
Wang et al [11]	✓	×	✓	×
CEP2U [17, 29]	✓	×	×	✓

Table 6: Cloud-friendly Optimization Characteristics per Approach

glects the potential for cloud deployment by not synthesizing parallel processing with adaptivity in query plans. On the other hand, few works include provisions for parallelism [12, 11, 17], geographically dispersed resource pools [13, 11] and the potential for multiple query optimization [13, 29]. What is more, is that despite the fact that [14, 27] are the sole approaches that include adaptivity provisions, they are deprived from parallelization properties. In what follows, we focus on how parallelism can be incorporated in CEP and in adaptivity/elasticity aspects that candidate CEP approaches should tackle upon targeting on deployments on cloud computing platforms.

6.1. Parallel CEP

Recent literature [36, 37, 12, 11] describes different strategies of parallelism tailored for CEP. Based on the ideas discussed in the aforementioned works, upon operating on cloud platforms, event query processing can take advantage of multiple processing nodes in one of the following ways:

– **Partition-based Parallelization.** Parallelism by partitioning the input event stream based on certain predicates is discussed in [36]. The generic concept is to use one or more attributes of incoming event tuples as the keys in a partition-map that assigns certain tuples to a processing unit. The partition-map ensures partition contiguity (see Section 2.3) and partition isolation during the extraction of pattern matches,i.e., by definition there is no need to synthesize partial matches between partitions. This kind of parallelism favors the adoption of SASE [10] inspired techniques and structures (such as PAIS) discussed in Section 3.1. Partition-based parallelization is a straightforward solution to incorporate partition contiguity as the chosen event selection strategy. Nevertheless, the adoption of an alternative event selection mode makes things more complicated. The reason is that full pattern matches are not isolated in a single partition anymore and an amount of partial matches would need to be continuously pipelined among processing nodes. Another important issue not

addressed in [36] is that the user-defined partition map and keys may need to be periodically adjusted to avoid unbalanced workload on certain processing units and under- or over-utilization upon changing data distributions.

– **State-based Parallelization.** State-based parallelization [37] entails that each processing unit is assigned a thread that is responsible for a certain FSM or NFA state. The processing is distributed in the sense that both buffer management as well as predicate-related optimizations (Sections 3.1, 4) concerning individual states are performed in each node individually. Partial pattern matches derived in each processing node that possesses tuple instances about a particular state in the FSM (or NFA), need to be pipelined to the node that processes tuples involving the next state in the FSM (or NFA) to extract full matches. An inherent limitation of pure (i.e. without combining it with other parallelization schemes) State-based parallelization is that the degree of parallelism cannot be configured. More precisely, it is only dictated by the number of event types incorporated in the pattern operators of the posed query. Moreover, partitioning incoming data according to the states in the FSM does not take into consideration event tuples' distribution thus being prone to load imbalance.

– **Operator-based Parallelization.** This strategy, used in [12], was discussed in Sections 3.2 and 5.1 as a representative approach. In a nutshell, Operator-based parallelism assigns operators to processing units while operators can send events to each other within a processing node and also to operators on other nodes so as to detect full pattern matches. Operator-based parallelism has similar drawbacks with the State-based approach.

– **Run-based Parallelization.** According to Run-based parallelism [37] processing is performed in batches of event tuples. Each batch includes ordered, i.e., pattern matching takes place on a given sequence of input events, tuples and possesses an identifier. Based on this identifier, it is assigned to a particular processing node where it initiates a run. To extract full matches that span multiple nodes, the end of a batch must be replicated at the beginning of the runs belonging to nodes receiving contiguous batches. Although the technique presented in [11] does not consider runs in batches of event tuples, its rationale resembles run-based parallelization. This is mainly due to the fact that it bases tuple assignment on (time) order of reception as well as it replicates part of the tuples received by a processing unit to nodes possessing contiguous relevant events. For fixed batch size, this approach is less prone to load imbalance. On the other hand, according to the techniques presented in [37], the part of a batch that needs to be replicated depends on the number, say n, of event types participating in the query operators. The degree of parallelization is also dependent on n since increasing the degree of parallelism for low n results in replicating entire batches in nodes receiving contiguous ones.

6.2. Towards Elasticity in CEP

Irrespectively of the parallelization strategy, opting for elasticity during processing complex events is not always an easy task. Here, we elaborate on the main issues that need to be tackled in order to achieve both elastic resource allocation and unhindered event query processing. Dynamic resource allocation requires *migration* to a

new query plan when mis-utilization is detected or QoS requirements [31] are violated. Nonetheless, upon deciding the new query plan expected to optimize elasticity related metrics [34], the actual migration process still needs to take place. Since the CEP system is already in action, previous (to migration) state needs to be handled with caution so as to avoid compromising the correctness of the CEP output. Previous state in the CEP setup mainly involves partial pattern matches that are processed in search of further expansion and event tuples included in the currently processed window based on the WITHIN clause of the query (Section 2.2).

The migration itself may be conducted in two alternative ways (adapted from [38] for a cloud setup): (a) in a stop-migrate-resume fashion or (b) on-the-fly. In the first case, the event processing tasks are stalled. Tuples in the current time window are replicated to the proper processing nodes based on the new plan. Also, previous partial matches are transfered to the nodes that are valid, given the chosen parallelization tactic, to investigate further expansion. Memory resources consumed by the old query plan are freed. Having prepared the deployment of the new query plan, query processing is restarted.

Pausing query processing for a while may not be acceptable for a variety of surveillance, monitoring or security applications which rely on continuous query processing and timely complex event detection to actuate decision making procedures. Therefore, for such CEP applications, on-the-fly migration is the only option. On-the-fly migration does not severely differ from what was described above. Nonetheless, instead of replicating state information in an offline fashion, on-the-fly migration initiates a parallel, totally new instance of the query processing. The input and output buffers of nodes executing the previous plan are shared between the old and new processing nodes with the old plan being progressively abandoned.

On-the-fly monitoring enables unhindered event query execution on par with elastic resource allocation on the counterweight of raising the workload and thus the costs of cloud usage. On the other hand, the stop-migrate-resume approach adds minimum overhead in resource utility, but implements elasticity by stopping query processing. This may be unacceptable for many application scenarios as it implies zero throughput during the migration process.

7. Predictive Analytics and CEP

Predictive Analytics (PA) refers to a wide range of methods for understanding data and discover knowledge out of them [39]. It has been subject of extensive research, especially in the fields of data mining and machine learning. In general, PA and CEP have been proven mutually beneficial [40]. For example, PA can be used to learn new complex events and rules from data [41, 42] or to learn predictive models based on complex events generated by a CEP engine [43]. Machine learning in particular has also been used to extract events from structured data, such as videos [44]. However, consistent with the focus of our study, in this section we concentrate on the synergies among CEP and PA approaches within the scope of event query optimization in both centralized and distributed setups. In particular, we discuss how PA can be useful in a CEP system to a) proactively determine when and how query plans should be adapted to avoid low throughput or mis-utilization, b) reduce communication costs in

distributed CEP settings. With respect to our discussion in the previous sections, the first of these issues is closely related to the adaptivity and elasticity concepts discussed in Section 3.2 and Section 6.2, while the second relates to distributed CEP settings studied in Section 5 and at the beginning of Section 6 (also see middle columns of Table 6).

When processing large, high velocity event streams, a crucial property of algorithms is scalability. In fact, if time or memory demands do not scale well with the number of tuples, candidate methods will fail to timely process potentially infinite streams and CEP system responsiveness will diminish over time. Methods that abide by these requirements are often termed online [45]. In PA, online model building maintains a model $w \in \mathcal{W}$ from an abstract model space $\mathcal{W}$. For each incoming event tuple $z_t \in \mathcal{Z}$ at timestamp t, the model is updated via an update rule:

$$\varphi : \mathcal{W} \times \mathcal{Z} \to \mathcal{W}$$
$$w_t = \varphi(w_{t-1}, z_t) \ .$$

Here, Z can be an arbitrary input space, such as the set of all possible event tuples.

When the model is used for prediction, these methods are instances of machine learning, where the most common tasks are the classification of event tuples or the prediction of an unknown function value based on the tuples, i.e., regression. In this case, the model represents a function that maps tuples to the target value. Machine learning with constant updates and constant time and space complexity is a well researched field, termed online learning [46].

When the task is instead to learn new rules or adapting rules to changes in the event distribution, online model building can be viewed as an instance of data mining. Here, the model usually comprises of patterns in the events, such as frequent event sets or subgroups. That is, a model can be regarded as a set of complex event patterns (or queries) that all abide a pre-defined interestingness criterion. Online adaptation of the model is difficult in this setting, because most criteria for event sets or subgroups to be interesting are defined in relation to the events that have been already observed. If the interestingness criterion can be linked to the frequency of single events, methods from online frequent itemset mining can be used [47]. Moreover, frequencies of events in a stream can be estimated using online kernel density estimation [48, 49]. *Regarding query optimization, online estimation of event frequencies can in particular be used to determine changes in the frequency distributions of events, enabling dynamic query planners to decide when and how existing query plans should be adapted.*

When applying these techniques in a distributed setting, information has to be exchanged from local sites to form a global model. Since centralization of events in Big Data environments is infeasible, either compressed versions of the event tuples or the local models themselves have to be exchanged. In practice, the model itself is orders of magnitude smaller than the set of observed event tuples, even when tuples are compressed. Thus, when possible, it is most beneficial to exchange only the model itself. For linear models, i.e., those that can be expressed as a finite-dimensional vector $w \in \mathbb{R}^d$, the most common practice is to combine local models to a global model by calculating their average. That is, the global model $w_t \in \mathbb{R}^d$ at time t, with respect to

$w_{t,l}$: $l \in [N]$, $N \in \mathbb{N}$ local models, is defined as:

$$w_t = \overline{w}_t = \frac{1}{N} \sum_{l=1}^{N} w_{t,l} \ .$$

In specific cases it can be shown that the average model is similar to a hypothetical centrally computed one [50]. However, in general no guarantee for the model quality can be given. Recently, a novel way of combining linear models has been suggested. The global model is defined as the Radon point [51] of the local models, where the Radon point is an approximation of the center point of the set of local models. It has been shown that this way of combining models can exponentially improve probabilistic guarantees of local models [51]. Given a sufficient number of local models, a global model generated by this approach maintains the quality of the hypothetical central one.

Exchanging only sketches of the tuples, event statistics or only the local models themselves reduces the communication overhead of distributed processing significantly. However, in what we discussed so far, information is exchanged for each incoming tuple. For high-velocity event streams, real-time services or applications on mobile devices, this amount of communication is still prohibitive. The state-of-the-art approach to this problem is to exchange information in mini-batches, i.e., only after a certain amount of events has been processed at each local site. In specific settings of online learning it has been shown that the reduced communication does not decrease the model quality severely. In particular, it has been shown that, in these settings, mini-batching retains asymptotic quality guarantees of centralized models [50].

Recently, a software framework has been released for communication-efficient distributed machine learning based on the idea of mini-batching [52]. A major drawback of this method is that it communicates even if the exchange of information does not improve the global model. A more data-driven strategy is to define a function that measures the usefulness of a model synchronization (i.e., the centralization of local models and their combination to a global model) and only synchronizes if the usefulness exceeds a user-defined threshold. There exist two major challenging issues that need to be confronted in such an approach: (a) a proper definition of a usefulness measure and (b) its communication-efficient monitoring. Usefulness measures may include information about the local models as well as local event tuples, thus a naive monitoring of such a measure would require the exchange of information. However, this would render using an adaptive model synchronization method pointless, because the communication saved by the adaptiveness is spend to monitor the usefulness function. Moreover, reasonable usefulness measures will seldom be linear in the input and monitoring non-linear functions over distributed data sources has been a major difficulty. In order to apply such an approach, each local site has to be able to monitor the global usefulness measures in a communication-efficient manner. With recent seminal advances on communication-efficient monitoring of non-linear functions [53, 54, 55] new methods of adaptive, communication-efficient model synchronization have been developed. To that extend, so far two usefulness measures have been proposed that both can be monitored communication-efficiently. We will describe both measures and outline how they can be monitored.

The first measure [56] has been developed based on the idea that, in order to decide

whether the global model should be updated, one has to decide if the data observed at each local site deviates significantly from the data used to generate the current global model. This idea has been applied to least squares regression, where the input space $\mathcal{Z} = \mathbb{R}^d \times \mathbb{R}$ consists of pairs (x_t, y_t), where $x_t \in \mathbb{R}$ is a feature vector and $y_t \in \mathbb{R}$ is the respective target value. E.g., x_t can be a binary vector indicating the presence of events in an event tuple. The model $w_t \in \mathbb{R}^d$ is again a real-valued vector and the task is to minimize the squared error over a collection of data items $E \subset \mathcal{Z}$, i.e.:

$$w^* = \arg\min_{w \in \mathbb{R}^d} \sum_{(x,y) \in E} \|xw - y\|_2^2 \ .$$

This optimization problem has a closed-form solution. Combining all feature vectors in a matrix $X \in \mathbb{R}^{|E| \times d}$ and target values in a vector $y \in \mathbb{R}^{|E|}$, we can express w^* as:

$$w^* = \left(X^\top X\right)^{-1} X^\top y \ .$$

Given a data window of size b, we set $E_t = \{(x_{t-b}, y_{t-b}), \ldots, (x_t, y_t))\}$ and X_t, y_t accordingly, so that

$$w_t = \varphi(X_t, y_t) = (X_t^\top X_t)^{-1} X_t^\top y_t \ .$$

Abbreviating $A = X^\top X, c = X^\top y$ yields $w = A^{-1}c$. Furthermore, A and c are time dependent and distributed over $N \in \mathbb{N}$ local sites so that at each point in time t we have $A_{t,1}, c_{t,1}, \ldots, A_{t,N}, c_{t,N}$ with

$$A_t = \frac{1}{N} \sum_{l \in [N]} A_{t,l} \ , \ \ c_t = \frac{1}{N} \sum_{l \in [N]} c_{t,l} \ .$$

The usefulness measure u now is defined as:

$$u_t = \frac{1}{N} \sum_{l \in [N]} \|A_{t'}^{-1} c_{t'} - A_{t,l}^{-1} c_{t,l}\| \ ,$$

where t' denotes the time of last synchronization. A synchronization is triggered, whenever $u_t > \epsilon$ for some threshold $\epsilon \in \mathbb{R}_+$. If $u_t \leq \epsilon$, this implies that also the difference between the last computed global model and the current one is below ϵ. This can be monitored communication-efficiently using the safe-zone approach [57], where each local site l monitors the following condition:

$$\|A_{t'}^{-1} c_{t'} - A_{t,l}^{-1} c_{t,l}\| \leq \epsilon \ .$$

Using this approach, the communication overhead for model synchronization could empirically be reduced by at least one order of magnitude [56].

The second approach [58] is based on the idea that model synchronizations are most beneficial if models are most diverse, and can be avoided when they are similar. It has been applied to online learning with linear models, where

$$w_t = \varphi(w_{t-1}, z_t) \ .$$

The global model $w_t \in \mathbb{R}^d$ is approximated by the average of the local models, i.e.:

$$w_t \approx \overline{w}_t = \frac{1}{N} \sum_{l \in [N]} w_{t,l} \ .$$

As mentioned above, for specific cases it has been shown that the average model is similar to the true global model (i.e., the hypothetical centrally computed one). Now the usefulness measure is defined as the variance amongst the models, i.e.:

$$u_t = \frac{1}{N} \sum_{l \in [N]} \|w_{t,l} - \overline{w}_t\|_2^2 \ .$$

As in the previous approach, a synchronization is triggered, whenever $u_t > \epsilon$ for some threshold $\epsilon \in \mathbb{R}$. Since $\overline{w}_t$ is unknown to the local sites without synchronization, again the safe-zone approach is applied. The local conditions that are monitored (without having to communicate) at each site are

$$\|w_{t,l} - \overline{w}_{t'}\|_2^2 \ ,$$

where t' again denotes the time of last synchronization. It has again been empirically shown that applying this technique, overhead communication can be reduced by an order of magnitude. Moreover, the communication overhead can be bounded by the number of local sites N and the loss bound, i.e., a measure of the hardness of the learning problem [45].

The impact of the results discussed in this section on CEP is two-fold. On the one hand, predictions obtained from such learning techniques enable proactive determination of query plan adaptation, for instance, when imminent resource mis-utilization is predicted. On the other hand, when applied within a CEP system to predict the upcoming of future events, predictive analysis can reduce transmission costs by allowing communication only when actual data yield complex events that deviate from what is predicted. Additionally, combining the potential for adaptivity and communication efficiency in distributed settings, approaches such as [56, 58] can be extended to learning and monitoring distributions over events. This enables data-driven dynamic query planning, where current query plans are altered as soon as the distribution over events—monitored locally at each site, but combined to a global model using the approaches discussed above—deviates from the distribution previously used to generate the query plan. Lastly, these novel approaches could be extended to other predictive analysis methods such as to communication efficient rule mining for scalable rule learning.

8. In Conclusion

Having reviewed the state-of-the-art CEP approaches focusing on efficient query processing and query optimization issues, we conclude our study summarizing our findings and point out interesting directions for future research. First, as our study reveals, despite the fact that centralized event processing in both the deterministic and uncertain contexts has been studied to some extend, especially from a data management

perspective in SASE [10, 24, 25, 19], important issues that arise in distributed settings have been surprisingly overlooked. This observation regards both distributed (parallel) processing and distributed monitoring over geographically dispersed sites, as pointed out in Section 5. More precisely:

– Although techniques for parallel CEP have been proposed in the literature, they are deprived from adaptivity considerations.

– Load balancing schemes for efficient resource manipulation and throughput maximization are absent from the discussion in related works.

– Existing techniques do not consider efficient processing of multiple queries combined with load balancing and adaptivity capabilities.

– Distributed monitoring techniques in the deterministic as well as probabilistic context are examined only in [13] and [11]. Nonetheless, [13, 11] view events as atomic units arriving at each site. Another meaningful event definition in distributed settings, not examined in the literature, regards complex events whose occurrence cannot be determined by a single site, but depends on the union of local event streams. For instance, in a simple form such an event may involve a global counter or sum [23] (comprised of local counters or sums in each site) exceeding a posed threshold.

The above exhibit that CEP over Big Data enabling technologies such as cloud platforms is still in an early stage. Efficient query processing over streaming event data in a cloud computing paradigm requires a constructive collaboration among just load balancing techniques, elastic (i.e. adaptive) resource allocation and efficient in situ processing schemes with provisions for synthesizing their final outcomes in the global context. In addition, many are the cases were application requirements for unhindered operation mandate the development of efficient ways for on-the-fly migration to new query plans, as part of the elastic resource allocation property. The need for such approaches is much more evident in cloud platforms with potentially dispersed resource pools apart from a number of proximate processing units.

PA and CEP synergies also open interesting directions towards:

– Data-driven, elastic query planning. Especially in cloud platforms, predicting misutilization or QoS standards' violation [31] can lead to proactive migration to efficient query plans.

– PA can reduce communication in distributed CEP platforms by allowing transmissions only when actual complex events occur beyond what is predicted.

All the above interesting directions are especially useful for the realization of CEP to planetary-scale Big Event Data. In fact, they constitute the focal points currently being studied in the context of the FERARI project [7] for which a multi-cloud CEP system [35] is being developed and enhanced.

[7] http://www.ferari-project.eu/

Acknowledgments

This work was supported by the European Commission under ICT-FP7-FERARI-619491 (Flexible Event pRocessing for big dAta aRchItectures).

References

[1] Y. Ahmad, B. Berg, U. Cetintemel, M. Humphrey, J.-H. Hwang, A. Jhingran, A. Maskey, O. Papaemmanouil, A. Rasin, N. Tatbul, W. Xing, Y. Xing, S. Zdonik, Distributed operation in the borealis stream processing engine, in: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, 2005, pp. 882–884.

[2] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, M. A. Shah, Telegraphcq: Continuous dataflow processing, in: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, 2003, pp. 668–668.

[3] S. Chakravarthy, Q. Jiang, Integrating stream and complex event processing, in: Stream Data Processing: A Quality of Service Perspective, Vol. 36 of Advances in Database Systems, Springer US, 2009, pp. 187–214.

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

[5] N. D. Doulamis, A. D. Doulamis, P. Kokkinos, E. Varvarigos, Event detection in twitter microblogging, IEEE Transactions on Cybernetics PP (99) (2015) 1–15.

[6] M. Zampoglou, A. G. Malamos, K. Kapetanakis, K. Kontakis, E. Sardis, G. Vafiadis, V. Moulos, A. Doulamis, ipromotion: A cloud-based platform for virtual reality internet advertising, in: Big Data and Internet of Things: A Roadmap for Smart Environments, Springer, 2014, pp. 447–470.

[7] A. Artikis, O. Etzion, Z. Feldman, F. Fournier, Event processing under uncertainty, in: Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems, DEBS '12, ACM, New York, NY, USA, 2012, pp. 32–43.

[8] E. Alevizos, A. Skarlatidis, A. Artikis, G. Paliouras, Complex event recognition under uncertainty: A short survey, in: Event Processing, Forecasting and Decision-Making in the Big Data Era (EPForDM), EDBT Workshop, 2015.

[9] I. Flouris, N. Giatrakos, M. Garofalakis, A. Deligiannakis, Issues in complex event processing systems, in: 1st IEEE International Workshop on Real Time Data Stream Analytics held in conjunction with IEEE BigDataSE-15, 2015.

[10] E. Wu, Y. Diao, S. Rizvi, High performance complex event processing over streams, in: Proceedings of SIGMOD, 2006.

[11] Y. Wang, K. Cao, X. Zhang, Complex event processing over distributed probabilistic event streams, Computers & Mathematics with Applications 66 (10) (2013) 1808 – 1821.

[12] N. P. Schultz-Moller, M. Migliavacca, P. Pietzuch, Distributed complex event processing with query rewriting, in: Proceedings of DEBS, 2009.

[13] M. Akdere, U. Cetintemel, N. Tatbul, Plan-based complex event detection across distributed sources, in: Proceedings of VLDB, 2008.

[14] Y. Mei, S. Madden, Zstream: A cost-based query processor for adaptively detecting composite events, in: Proceedings of SIGMOD, 2009.

[15] S. Wasserkrug, A. Gal, O. Etzion, Y. Turchin, Complex event processing over uncertain data, in: Proceedings of the Second International Conference on Distributed Event-based Systems, DEBS '08, ACM, New York, NY, USA, 2008, pp. 253–264.

[16] C. Ré, J. Letchner, M. Balazinksa, D. Suciu, Event queries on correlated probabilistic streams, in: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD '08, ACM, New York, NY, USA, 2008, pp. 715–728.

[17] G. Cugola, A. Margara, M. Matteucci, G. Tamburrelli, Introducing uncertainty in complex event processing: model, implementation, and validation, Computing (2014) 1–42.

[18] Z. Shen, H. Kawashima, H. Kitagama, Probabilistic event stream processing with lineage, in: Proceedings of Data Engineering Workshop (DEWS), 2008.

[19] H. Zhang, Y. Diao, N. Immerman, Recognizing patterns in streams with imprecise timestamps, Proc. VLDB Endow. 3 (1-2) (2010) 244–255.

[20] X. Chuanfei, L. Shukuan, W. Lei, Q. Jianzhong, Complex event detection in probabilistic stream, in: Web Conference (APWEB), 2010 12th International Asia-Pacific, 2010, pp. 361–363.

[21] S. Wasserkrug, A. Gal, O. Etzion, A model for reasoning with uncertain rules in event composition systems, CoRR abs/1207.1427.
URL http://arxiv.org/abs/1207.1427

[22] S. Wasserkrug, A. Gal, O. Etzion, Y. Turchin, Efficient processing of uncertain events in rule-based systems, Knowledge and Data Engineering, IEEE Transactions on 24 (1) (2012) 45–58.

[23] O. Etzion, P. Niblet, Event Processing in Action, Manning Publications Co, 2011.

[24] H. Zhang, Y. Diao, N. Immerman, Optimizing expensive queries in complex event processing, in: Proceedings of SIGMOD, 2014, pp. 217–228.

[25] J. Agrawal, Y. Diao, D. Gyllstrom, N. Immerman, Efficient pattern matching over event streams, in: Proceedings of SIGMOD, 2008.

[26] R. S. Barga, J. Goldstein, M. Ali, M. Hong, Consistent streaming through time: A vision for event stream processing, in: Proceedings of 3rd Biennial Conference on Innovative DataSystems Research (CIDR), 2007.

[27] I. Kolchinsky, I. Sharfman, A. Schuster, Lazy evaluation methods for detecting complex events, in: Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems, DEBS '15, 2015, pp. 34–45.

[28] E. Rabinovich, O. Etzion, A. Gal, Pattern rewritting framework for event processing optimization, in: Proceedings of DEBS, 2011.

[29] G. Cugola, A. Margara, Complex event processing with t-rex, J. Syst. Softw. 85 (8) (2012) 1709–1728.

[30] A. Demeers, J. Gehrke, B. Panda, M. Riedewald, V. Sharma, W. White, Cayuga: A general purpose event monitoring system, in: Proceedings of 3rd Biennial Conference on Innovative DataSystems Research (CIDR), 2007.

[31] N. D. Doulamis, P. Kokkinos, E. Varvarigos, Resource selection for tasks with time requirements using spectral clustering, IEEE Transactions on Computers 63 (2) (2014) 461–474.

[32] Y. Zhou, C. Ma, Q. Guo, L. Shou, G. Chen, Sequence pattern matching over event data with temporal uncertainty, in: Proc. 17th International Conference on Extending Database Technology (EDBT), Athens, Greece, March 24-28, 2014., 2014, pp. 205–216.

[33] D. Calvanese, G. De Giacomo, M. Lenzerini, M. Y. Vardi, Springer Berlin Heidelberg, 2000, Ch. What Is Query Rewriting?, pp. 51–59.

[34] S. Lehrig, H. Eikerling, S. Becker, Scalability, elasticity, and efficiency in cloud computing: A systematic literature review of definitions and metrics, in: Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures, 2015, pp. 83–92.

[35] I. Flouris, V. Manikaki, N. Giatrakos, A. Deligiannakis, M. Garofalakis, et al., FERARI: A Prototype for Complex Event Processing over Streaming Multi-cloud Platforms, in: SIGMOD, 2016 (to appear).

[36] M. Hirzel, Partition and compose: Parallel complex event processing, in: Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems, 2012, pp. 191–200.

[37] C. Balkesen, N. Dindar, M. Wetter, N. Tatbul, Rip: Run-based intra-query parallelism for scalable complex event processing, in: Proceedings of the 7th ACM International Conference on Distributed Event-based Systems, 2013, pp. 3–14.

[38] Y. Zhu, E. A. Rundensteiner, G. T. Heineman, Dynamic plan migration for continuous queries over data streams, in: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, SIGMOD '04, ACM, New York, NY, USA, 2004, pp. 431–442.

[39] E. Siegel, Predictive analytics: The power to predict who will click, buy, lie, or die, John Wiley & Sons, 2013.

[40] G. Tóth, L. J. Fülöp, L. Vidács, A. Beszédes, H. Demeter, L. Farkas, Complex event processing synergies with predictive analytics, in: Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems, 2010, pp. 95–96.

[41] L. J. Fülöp, Á. Beszédes, G. Tóth, H. Demeter, L. Vidács, L. Farkas, Predictive complex event processing: a conceptual framework for combining complex event processing and predictive analytics, in: Proceedings of the Fifth Balkan Conference in Informatics, ACM, 2012, pp. 26–31.

[42] D. Seipel, P. Neubeck, S. Köhler, M. Atzmueller, Mining complex event patterns in computer networks, in: New Frontiers in Mining Complex Patterns, Springer, 2013, pp. 33–48.

[43] M. Miwa, R. Sætre, J.-D. Kim, J. Tsujii, Event extraction with complex event classification using rich features, Journal of bioinformatics and computational biology 8 (01) (2010) 131–146.

[44] K. Tang, L. Fei-Fei, D. Koller, Learning latent temporal structure for complex event detection, in: Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on, IEEE, 2012, pp. 1250–1257.

[45] M. Kamp, M. Boley, M. Mock, D. Keren, A. Schuster, I. Sharfman, Adaptive communication bounds for distributed online learning, in: Proceedings of the 7th NIPS Workshop on Optimization for Machine Learning, 2014.

[46] T. Anderson, The theory and practice of online learning, Athabasca University Press, 2008.

[47] J. H. Chang, W. S. Lee, Finding recent frequent itemsets adaptively over online data streams, in: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2003, pp. 487–492.

[48] M. Kristan, A. Leonardis, D. Skočaj, Multivariate online kernel density estimation with gaussian kernels, Pattern Recognition 44 (10) (2011) 2630–2642.

[49] F. Desobry, M. Davy, C. Doncarli, An online kernel change detection algorithm, Signal Processing, IEEE Transactions on 53 (8) (2005) 2961–2974.

[50] O. Dekel, R. Gilad-Bachrach, O. Shamir, L. Xiao, Optimal distributed online prediction using mini-batches, The Journal of Machine Learning Research 13 (1) (2012) 165–202.

[51] T. Gärtner, O. Missura, Online optimisation in convexity spaces, in: Proceedings of the NIPS Workshop on Discrete and Combinatorial Problems in Machine Learning (DISCML), 2014.

[52] M. Li, D. G. Andersen, A. J. Smola, K. Yu, Communication efficient distributed machine learning with the parameter server, in: Advances in Neural Information Processing Systems, 2014, pp. 19–27.

[53] I. Sharfman, A. Schuster, D. Keren, A geometric approach to monitoring threshold functions over distributed data streams, ACM Transactions on Database Systems (TODS) 32 (4) (2007) 23.

[54] N. Giatrakos, A. Deligiannakis, M. Garofalakis, I. Sharfman, A. Schuster, Prediction-based geometric monitoring over distributed data streams, in: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, ACM, 2012, pp. 265–276.

[55] N. Giatrakos, A. Deligiannakis, M. Garofalakis, I. Sharfman, A. Schuster, Distributed geometric query monitoring using prediction models, ACM Trans. Database Syst. 39 (2) (2014) 16:1–16:42.

[56] M. Gabel, D. Keren, A. Schuster, Monitoring least squares models of distributed streams, in: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2015, pp. 319–328.

[57] D. Keren, I. Sharfman, A. Schuster, A. Livne, Shape sensitive geometric monitoring, Knowledge and Data Engineering, IEEE Transactions on 24 (8) (2012) 1520–1535.

[58] M. Kamp, M. Boley, D. Keren, A. Schuster, I. Sharfman, Communication-efficient distributed online prediction by dynamic model synchronization, in: T. Calders, F. Esposito, E. HG?llermeier, R. Meo (Eds.), Machine Learning and Knowledge Discovery in Databases, Vol. 8724 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2014, pp. 623–639.