Recent Advancements in Event Processing

MIYURU DAYARATHNA and SRINATH PERERA, WSO2 Inc.

Event processing (EP) is a data processing technology that conducts online processing of event information. In this survey, we summarize the latest cutting-edge work done on EP from both industrial and academic research community viewpoints. We divide the entire field of EP into three subareas: EP system architectures, EP use cases, and EP open research topics. Then we deep dive into the details of each subsection. We investigate the system architecture characteristics of novel EP platforms, such as Apache Storm, Apache Spark, and Apache Flink. We found significant advancements made on novel application areas, such as the Internet of Things; streaming machine learning (ML); and processing of complex data types such as text, video data streams, and graphs. Furthermore, there has been significant body of contributions made on event ordering, system scalability, development of EP languages and exploration of use of heterogeneous devices for EP, which we investigate in the latter half of this article. Through our study, we found key areas that require significant attention from the EP community, such as Streaming ML, EP system benchmarking, and graph stream processing.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Information systems** → **Data streams**; **Stream management**; **Data streaming**; • **Theory of computation** → *Streaming, sublinear and near linear time algorithms*;

Additional Key Words and Phrases: Event processing, complex event processing, data stream processing

ACM Reference format:
Miyuru Dayarathna and Srinath Perera. 2018. Recent Advancements in Event Processing. *ACM Comput. Surv.* 51, 2, Article 33 (February 2018), 36 pages.
https://doi.org/10.1145/3170432

1 INTRODUCTION

Event processing (EP) is a paradigm where streams of events are analyzed to extract useful insights of real-world events. Several real-world applications can be found with the requirement of processing information flowing from outside to the system [48]. A vast number of recent applications of EP could be found in areas such as health informatics [31], astronomy [127], telecommunications [184], electric grids [89], energy [95], geography, and transportation [105]. These applications need to process large amounts of data generated by various different sensors online satisfying soft real-time constraints. Since the amount of data generated by these sensors are tremendous, it is impossible to follow a store and process technique with processing such data. Most of the time, the data processing conducted by such applications are lightweight one-time operations such as filtering and transformation but may also involve joining multiple streams and different types of windows, for example. A subset of EP applications (Figure 1) may involve detecting complex events, such as anomaly detection and predictive analytics, which are termed *complex event processing* (CEP).

Authors' addresses: M. Dayarathna and S. Perera, WSO2, Inc., 787, Castro Street, Mountain View, CA, USA 94041; email: {miyurud, srinath}@wso2.com.

https://doi.org/10.1145/3170432

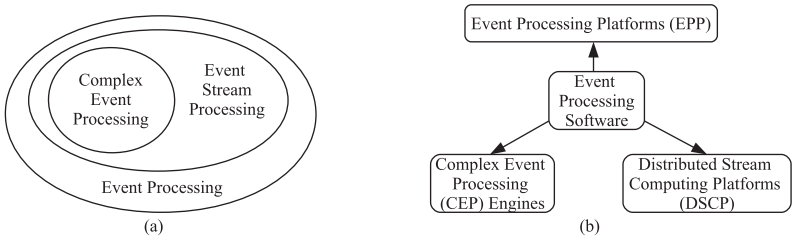

Fig. 1. Event processing landscape. (a) Difference between EP terminologies. (b) Types of EP software. In this article, we classify the entire domain of EP software into three areas: CEP engines, EPPs, and DSCPs.

EP has gained a significant market share in data processing technologies. Two recent articles forecast that the CEP market will grow from $1,005 million to $4,762 million in 2019 [2], whereas streaming analytics market will grow from $502.1 million in 2015 to $1,995.7 million by 2020 [4]. These articles report compound annual growth rates (CAGRs) of 31.3% and 36.5%, respectively.

The roots of EP can be traced back to the era of active databases [7]. Since then, several EP software systems have appeared. EP can be divided into two main areas: stream processing and CEP. The first area, stream (i.e., event) processing, supports many kinds of continuous analytics, such as filter, aggregation, enrichment, classification, and joining. CEP, however, uses patterns over sequences of simple events to detect and report composite events. The boundaries between CEP and stream processing are not always clear. In certain works, CEP has been implemented as an operator in a general-purpose stream processing system [97]. However, CEP engines have implemented support for conducting general-purpose EP. We believe that CEP can be considered as a special form of EP, and hence the theories and applications devised for EP are equally applicable to CEP as well (see Figure 1(a)). A recent article by Schulte [166] categorized EP systems as event stream processing (ESP) platforms, event processing platforms (EPPs), distributed stream computing platforms (DSCPs), and CEP platforms or engines. In this article, we adapt the notion in categorizing EP software systems described in [166] (see Figure 1(b)).

Although a few surveys have been conducted on EP systems, with the most recent one published almost half a decade ago, a significant body of research has been conducted in the realm of EP systems. New DSCPs have been introduced recently. New use cases of EP have arrived, and technology advancements have been made relating to the quality-of-service (QoS) characteristics of EP systems. However, no recent survey has been conducted on these rising trends in EP.

The aim of this survey is to summarize the significant recent industry initiatives and research efforts made on EP and build on them to forecast on significant research directions worth of pursuing. We first provide an introduction to the area of EP and then describe the framework followed in our survey. We categorize the EP products based on their important features. Finally, we describe the areas that are significant venues for further innovations, including Internet of Things (IoT) initiatives, edge analytics, visual query interfaces, streaming machine learning (ML), autotuning of EP query networks, EP provenance, EP benchmarking efforts, and EP query languages.

2 RELATED SURVEYS

There have been few detailed surveys conducted in the context of EP systems. The surveys published until now can be categorized into three main areas: EP in general, streaming algorithms, and stream processing optimizations.

Similar to our work, the majority of existing surveys on EP have been on EP in general settings. One of the earliest surveys on general EP topics was conducted by Fülöp et al. [71], who examined

terminology, research achievements, existing solutions, and open issues of CEP. They provided an introduction to innovative works on CEP, basic concepts of CEP, and a detailed taxonomy of CEP tools. Furthermore, they conducted a detailed assessment of the technologies of predictive analytics. The survey made around 2012 by Cugola et al. [48] unified data stream processing and CEP under a single term: *information flow processing* (IFP). They claimed that neither data stream processing nor CEP models could completely function as an IFP engine that consists of characteristics of the two areas. Similar to us, they drew a framework that is general enough for comparisons. They used this framework and made a comprehensive investigation of the cutting-edge research. However, the field of EP has undergone significant developments since the publishing of Cugola et al.'s work [48]. For example, the field of the second-generation distributed stream processing systems (DSCPs) was developed within the past 5 to 7 years. Furthermore, novel application areas of EP have emerged, such as IoT and streaming ML. Hence, we believe that it is important to investigate these recent trends in EP. In a more recent survey, Flouris et al. [69] made an overview of the main research issues faced by existing CEP technologies. They created a taxonomy of CEP performance optimization approaches highlighting the main areas, such as predicate-related optimizations, buffering techniques, and query rewriting/reordering. Furthermore, they discussed adaptive CEP (i.e., alter query execution plan at runtime), CEP in distributed settings, and CEP with imprecisions in the data. However, our survey discusses recent advancements of EP in depth by classifying the broad area of EP into three main areas of focus: EP use cases, EP system architectures, and EP open research topics (see Figure 1(b)). In another recent survey, Sun et al. [183] investigated on big data stream processing (BDSP) from the point of view of the systems used for BDSP. In their work, they first provided a general overview of the BDSP system architecture. Next, they described the system architecture aspects of Storm [189], S4 [147], TimeStream [153], and Naiad [143] systems. The survey made by Gaber et al. [72] concerned stream processing on ubiquitous environments. Different from both of these works, we conduct the survey with much more broader scope.

A few notable surveys were conducted on the algorithmic aspects of stream processing. Stream clustering has been an area of great importance. The work done by Silva et al. [170] is one of the earliest examples of this category of surveys. A second example is the work done by Aggarwal [11]. The work by Amini et al. [17] is the third example of such surveys. Graph stream algorithms is another category of stream processing algorithms that has attracted significant attention. The survey published by McGregor [133] highlights multiple works conducted in this area. However, in this article, we survey the area from the perspective of graph stream processing applications.

The work by Hirzel et al. [100] is an example of a survey conducted on the performance optimization techniques of data stream processing. They created a comprehensive catalog of stream processing optimizations. We highlight only a few important techniques in this article, ours is on general data stream processing topics.

A chronologically ordered list of the aforementioned surveys is presented in Table 1. In this article, we focus on EP in general and discuss the latest trends in EP. We believe that the comprehensive coverage made on the latest state-of-the-art research on EP will be of great interest to the EP community.

3 OVERVIEW OF THE EVENT PROCESSING

An event is simply a real-world incident that has been captured by a system. A complex event is an event derived from a group of events using either aggregation of derivation functions [65]. Information enclosed in a set of related events can be represented (i.e., summarized) through such a complex event. EP is the computing that captures and processes the occurrence of real-world incidents. CEP can be defined as computing that conducts operations on complex events. [166].

Table 1. Comparison of Related Surveys

Year	Investigator(s)	Area of Focus
2010	Fülöp et al. [71]	Comparison of CEP tools and their application to predictive analytics
2012	Cugola and Margara [48]	General survey of CEP and data stream processing
2013	Aggarwal [11] and Silva et al. [170]	Stream clustering
2014	Cabanillas et al. [5]	Stream processing systems and languages
2014	Amini et al. [17]	Stream clustering
2014	Hirzel et al. [100]	Stream processing optimization
2014	Gaber et al. [72]	Stream processing in mobile environments.
2014	McGregor [133]	Graph stream algorithms
2015	Flouris et al. [69]	General survey of CEP systems
2015	Sun et al. [183]	Stream computing systems

Events of different formats are gathered from different event producers. The event producers can be of various different types, including financial feeds, news feeds, weather sensors, application logs, and video streams collected from surveillance cameras. The EP engine is the brain of the processing that does multiple types of processing on event streams based on predefined rules. The processing includes simple counting, averaging, and median calculation type of simple EP operations, as well as more complex processing such as pattern matching and event prediction (forecasting). Event consumers are parties who are interested in mining valuable information from the event streams, such as software agents and users of Web/mobile applications.

In this article, we divide the advancements made on EP into three subcategories: EP system architectures, EP use cases, and open research topics on EP. EP system architectures are advancements made on the architectural aspects of EP systems. EP use cases are novel application domains in which the EP has become prominent in recent years. IoT data stream processing with complex data types such as text, video, and graphs are some examples in this category. In this article, we describe recent EP use cases from the point of view of general technological paradigms. For example, although video stream analysis can be conducted for security purposes or environmental monitoring, we concentrate on the general technologies of video stream processing without listing the specific real-world implementations of such technologies. A survey on specific real-world implementations is available from Hinze et al. [96]. We categorize the improvements made on the operational aspects of EP systems such as high availability (HA) and fault tolerance (FT), security, correctness, performance, and scalability under EP open research topics. It should be noted that although we have made the three preceding categories, in a particular advancement made on EP, several of these categories can be interlinked together.

4 EVENT PROCESSING SYSTEM ARCHITECTURES

The implementation of EP concepts is made in the context of ESP systems. The software tools used for EP can be categorized under three subtypes: EPPs [166], DSCPs, and CEP libraries (i.e., CEP engines). This section elaborates more on these subtopics. EPPs are types of ESP software that provide high-level programming models such as expressive event processing languages (EPLs) and built-in functions for event filtering, correlation, and abstraction. DSCPs (which also can be called *distributed stream processing engines* (DSPEs) [19]) are ESP platforms that provide explicit support

for distribution of computation across multiple nodes in a compute cluster. The CEP library (we also use the term *CEP engine* interchangeably) in general is a software component that specially focuses on detection of complex events.

4.1 Event Processing Platforms

IBM InfoSphere Streams [24], DataTorrent Real-Time Streaming (RTS) [53] (Apache Apex), SQLStream s-Server [177], WSO2 CEP [105], and StreamMine3G [131] are some notable examples of EPPs. Note that EPPs can be developed following a publish/subscribe pattern. In such systems, events generally are delivered to interested parties via a message broker (i.e., notification service [142]). Message brokers can be configured to filter out certain unwanted events so that further processing can be done on the pure EP system.

A summary of feature comparisons between different EPPs is shown in Table 2. Note that if a feature is present, we mark it with a black bullet (●), whereas the missing features are marked with a dash (—). If the details of a specific feature are not known (e.g., the details are unavailable), we keep the cell in an empty status. In terms of SQL-like query languages, a few EP systems, such as TIBCO StreamBase, WSO2 CEP, and SQLstream, support this language. Streams processing language (SPL) of IBM InfoSphere Streams is a dataflow language with different syntax from SQL. SPL is a streaming language; it is not an EP language, as it does not primarily concentrate on matching patterns over event streams [98]. All except the Fujitsu Interstage CEP provides query composition wizards and distributed processing functionalities. Features such as DB integration, JMS integration, and visual application debugging are supported by most of the EP systems listed in Table 2. Note that in the table, we list four types of windows: sliding windows, time windows, tuple windows, and batch windows. The sliding window expires only part of the events stored in the window. The time window completely expires the content of the windows when a predefined elapsed time passes. The tuple window expires its content upon receipt of a punctuation tuple. The batch window (also known as the tumbling window) expires its entire content at the same time. Note that the scalability results of the DSCPs listed in Table 2 are indicated in Tibbetts [185] (TIBCO StreamBase), Nabi et al. [145] (IBM InfoSphereStreams), Ravindra and Dayarathna [157] (WSO2 CEP), and SQLstream [176] (SQLstream). An essential summary of the EPP feature comparison is given in Table 2, but extended versions of such comparisons can be found in Luckham [125] and Vincent [194].

4.2 Distributed Stream Computing Platforms

There are clearly distinguishable categories of DSCPs based on their architecture. We classify these systems as first-generation, second-generation, and geodistributed DSCPs.

The first-generation DSCPs enforce a relational view [135] on their users mainly due to their lineage with the relational database systems. Borealis is an example of this category of systems [6]. The design of Borealis emphasized three fundamental requirements for stream processing engines (SPEs), such as (1) dynamic revision of query results, (2) dynamic query modification, and (3) flexible and highly scalable optimization.

However, we have witnessed the second generation of streaming systems that essentially do not enforce a relational view on their users. The notable examples are S4 [147] and Storm [189]. There are two main variations of second-generation DSCPs: per-event processing and microbatching (i.e., discretized/batched stream processing) [192]. Per-event processing EP systems treat each and every event that they receive individually. They provide very low latencies while low throughput compared to their counterpart (microbatching). Example systems that implement per-event processing include S4, Storm, and Flink. A new distributed data processing paradigm called *batched stream processing* has made a paradigm shift in the EP system architectures in recent times. These

Table 2. Feature Comparison of Event Processing Platforms

Feature	TIBCO Stream-Base	IBM Infosphere Streams	DataTorrent RTS	Amazon Kinesis	WSO2 CEP	Fujitsu Interstage CEP	SQLstream
SQL-like queries	●				●		●
Query composition	●	●	●	●	●		●
Temporal Pattern and sequence queries	●		●		●	●	
Key query types: Joins (J), Windows (W)	J/W	J/W	J/W		J/W		J/W
Window types: Sliding (S), Time (I), Tuple (T), Batch (B)	S	S / I / T / B	S / I / B		S / I / T[3] / B		S / I / B
Integrate with GIS Servers?	—	—				●	
Show results	●	●	●	●			●
DB Integration	●	●	●		●	●	●
Distributed Processing	●	●	●	●	●		●
Visual Application Debugger?	●	●	●		●		●
Messaging system support: JMS (M)/ Kafka (K)	M/K	M	M/K		M/K	M	M/K
Built-in Fault Tolerance[2]		●	●	●[4]			
High availability	●	●	●	●	●		
Scalable?	●	●		?[5]	●		?[5]
Best reported performance	91,000 evals/s [186]	7.3 Million Tuples/s [132]	76 Million tuples/s (moving average) [158]	—	0.4 Million events/s [151]	upto 2 Million events/s [70]	4.68 Million records/s [178]
Comprehensive operator library	●	●	●		●		●
API support/ language extensions for geoinformation processing	—	—		●	●	●	
Open source license: Apache (A), GNU GPL (G), BSD License (B)	—	—	A[1]	—	A	—	—

[1] DataTorrent RTS core engine is available under Apache 2.0 License as Apache Apex [159].
[2] The difference between FT and HA is described in Section 6.8.
[3] In WSO2 CEP Tuple windows are termed as *Unique Window*.
[4] Via Amazon Kinesis Client Library.
[5] Performance characterizations have been made without any scalability analysis [28, 176].

Table 3. Feature Comparison of Distributed Stream Computing Platforms

Feature	S4	Storm	Spark Streaming	Samza	Apex	Flink
SQL-like queries		●	●			●
DB Connection	●	●	●	●	●	●
Relational DB	●	●	●	●	●	●
Key query types: Joins (J), Windows (W)	J, W	J	J, W	W	J, W	J, W
Window types: Sliding (S), Time (I), Tuple (T), Batch (B)	S	—	S, I, T, B	I	S, I, T, B	S, I, T, B
Fault tolerance	—	Exactly-once	Recompute	State persistence	State persistence	Exactly once/at least once
HA	●	●	●[1]	●	●	●
Scalable?[2]	●	●	●		●	●
Best reported performance	Up to 10 thousand events/s [58]	3.2 Million tuples/s [146]	130,000 events/s [162]	1.2 Million messages/s [68]	3.3 Million tuples/s [159]	15 Million events/s [85]

[1] HA in Spark Streaming has been implemented using Apache Mesos.
[2] Level of scalability significantly differs from each DSCP.

systems provide high throughput but introduce relatively high latency. Comet was one of the first systems to implement this technique, and it was integrated with DryadLINQ [90]. Spark Streaming and Storm Trident are examples for systems that follow microbatching.

Spark Streaming [200], Samza [21], Apache Flink (previously Stratosphere) [20], Trill [41], and Apache Apex [159] are examples of DSCPs that have both batch and stream processing capabilities. They have resulted from efforts made to develop unified processing frameworks for big and fast data. Most of these systems allow the users to specify an expected level of latency while maximizing the system's throughput [121]. These systems cannot be directly categorized as EPPs as such but can be used to implement EP functionality.

Geodistributed stream processing has become an important topic in recent times due to the prevalence of cost-efficient data centers around the globe [191]. General performance metrics of EP systems such as latency, throughput, and performance variability become significant factors in operating a geodistributed EP installation [191]. Conducting high-performance ESP across data centers requires significant changes from the conventional EP system architectures. To create an efficient event transfer, batched event transferring has to be adapted in such systems. There are two key challenges on event batching: the size of the event batch and when to trigger sending the batch. Latency models can be created to support dynamic adaptation of the batch size to the cloud environment and to enable multiroute streaming across cloud nodes.

DSCPs are still evolving in various aspects, such as declarative query languages and FT. DSCPs such as Storm have undergone significant architecture enhancements in recent times [118, 189]. A summary of the feature comparison of some of the second-generation DSCPs is shown in Table 3. Currently, only Spark Streaming, Storm, and Flink are equipped with an SQL-like query language. All of the DSCPs listed in Table 3 are released under the Apache 2.0 license. The key query types

Table 4. Feature Comparison of Complex Event Processing Libraries

Feature	Esper (Basic Version)	Siddhi	RuleCore	Cayuga
SQL-like queries	●	●	●	
DB connection	●	●		
Open source license: Apache (A), GNU GPL (G), BSD License (B)	G	A	G	B
Debugging support	●			
Window types: Sliding (S), Time (I), Tuple (T), Batch (B)	I, T, B	S, I, T, B	T, S	T, S
sequences	●	●	●	
Event Aggregation	●	●		●
Nested queries	●			
Data extraction	●	●		
Parameterization	●	●		
Best-reported performance	500 000 event/s [63]	8.55 Million events/s [150]	—	—

and window types shown in Table 3 correspond to the support (explicit API support) for implementing the main streaming query categories with the specified DSCP. For example, although stream joins can be implemented using Apache Samza, its API does not include explicit support for implementing a join operation.

4.3 Complex Event Processing Libraries

The CEP library should be able to identify meaningful patterns, relationships, and data abstractions among apparently unrelated events, and fire a response immediately. CEP libraries are lightweight components. They can be embedded in other EP systems, such as in DSCPs and EPPs.

Several notable CEP libraries have appeared recently in both industry and academia. Esper (basic version) [64], Siddhi [181], ruleCore [160], and Cayuga [34] are some of the notable CEP libraries that have appeared recently. A comparison of the features of CEP libraries is shown in Table 4. The dots on each cell indicate the presence of the designated feature on the CEP library. Note that we follow the same convention throughout the rest of the article.

All of the CEP libraries listed on Table 4 are following an open source license scheme. Furthermore, none of them currently supports geospatial data types [5]. We conducted an analysis on the technologies used for programming modern open source EP systems. We have listed the percentages of programming languages used by a selected list of open source EP systems in Figure 2. We found that most of the systems have been implemented using Java (52%). Another 15% of the system architectures have been implemented with Scala. For example, 78% of Spark and 56% of Samza have been implemented completely in Scala. However, few if not any of the compared CEP systems have been developed in languages such as C/C++ and C#.

5 EVENT PROCESSING USE CASES

The EP paradigm has undergone rapid growth in recent years. Out of the number of topics investigated over the years, several areas of research have attracted significant attention. In this section, we summarize the most notable recent use cases of EP.

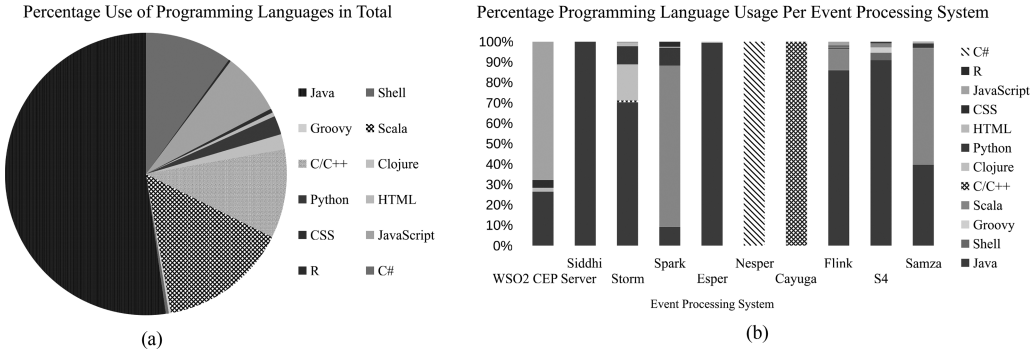

Fig. 2. EP software programming language usage. The pie chart on the left-hand side presents an aggregated view, whereas the right-hand side bar chart shows a breakdown of the programming languages used in different EP systems.

5.1 Internet of Things Initiatives

The IoT has become a hot topic in the EP paradigm. Data generated by IoT devices are streaming data, and often they are time series data as well. Most IoT use cases need real-time results, which requires streaming analysis. Typical IoT analytics can be conducted at three levels: single device–level analytics, analytics across multiple devices, and predictive analytics. Single device–level analytics focuses on allowing users to monitor the device's statistics and receive alerts about the device. Interesting device-level statistics could be obtained from an EP engine. The use of analytics conducted across multiple devices allows for aggregating data across different levels of the device hierarchy and let users post custom queries on the aggregated data. Predictive analytics (e.g., classification, predicting the next value, anomaly detection) enables selection of a set of data and conduction of advanced analysis using techniques such as ML. Applications of EP technologies can be found on all of these three levels. For example, Open Mobile miner can be used to build lightweight mobile applications that run in single device, such as mobile activity recognition [72]. Coll-stream is an example of a stream processing application that involves multiple devices [82].

Recently, there have been multiple interesting use cases of EP in IoT-related research. One such example is the automotive embedded data stream management system (AEDSMS) [199]. The embedded systems installed within a vehicle are used to support autonomous and safe driving. A new breed of challenges is faced when designing AEDSMS, such as query precompilation, distributed stream processing using in-vehicle networks (networks within the vehicle), real-time scheduling, and sensor data fusion. General DSMSs allows for registering, removing, and modifying queries as necessary on the fly during operation. However, automotive embedded systems (i.e., AEDSMS) need to satisfy real-time requirements to avoid accidents. The ability to predict worst-case latency is a very important requirement in this regard. During the automotive development, hardware configurations and application software are determined at the design stage. Hence, the data processing that gets installed in the automotive embedded systems are also determined at that time. Therefore, query structures must be determined at the design stage in the automotive field. Such stringent requirements make the task of designing an AEDSMS complicated.

Smart city applications are another example of IoT initiatives. Within buildings, there are multiple subsystems, such as power generators, heating and ventilation units, AC, switch gears, and elevators. Smart city applications extract real-time information from such things and run analytics, and provide useful information that could be used for saving energy, providing comfortable environment, and remote maintenance, among others. Smart city applications are hindered by various

issues, such as difficulty in discovering the capabilities of the available infrastructure, integrating heterogeneous data sources, and extracting up-to-date information in real time. Ontologies have been created to assist with describing complex event services. In one such initiative, Gao et al. [75] presented the automated complex event implementation system (ACEIS), which is an automated discovery and integration system for urban data streams. It receives requirements from users and applications as event requests and discovers or composes the relevant data streams to address the requirements specified in the event requests. IoT vendors such as Pacific Controls have developed 24/7 remote monitoring and control systems to monitor light data that come in from the connected buildings.

Another example of an IoT EP scenario can be found in the enterprise communications domain. The work by Ali et al. [15] presented a conceptual architecture of IoT-enabled communication systems. IoT applications involving EP with smart grid data have appeared as well [206]. One such use case is the warning of a peak power load by processing streaming data from smart meters [171].

5.1.1 Edge Analytics. Edge analytics (also known as fog computing [44]) is an emerging topic in EP software systems that is closely related to IoT. The International Data Corporation (IDC) predicts that by 2018, 40% of IoT-created data will be stored, processed, analyzed, and acted upon close to or at the edge of the network [155]. The problem is motivated by the size of the IoT data, which can range from gigabytes to terabytes per day, and the lack of a high bandwidth wide area network connecting IoT deployments where at least some connections are wireless. The key idea in edge analytics is to reduce the large amount of data that otherwise would have to be streamed from the event producer to the CEP engine. Edge analytics conducts partial EP near event sources (inside the devices where the data gets generated) or at the gateway (e.g., Dell Edge Gateway 5000 Series [3]). It transfers the processed information (small in size) from those edge analytics devices to the central location where decision making happens.

Edge analytics has significant potential in future EP initiatives since the amount of big data generated from different streaming sources continues to grow exponentially. The ParStream IoT analytics platform (shown in Figure 3) is one example of an industrial geodistributed edge analytics platform [30]. SQL queries are received by ParStream from a variety of analytical applications that get connected to it. The ParStream query is broken down into a series of smaller SQL queries for each of the nodes. Each of these subqueries, along with the central data (these are small tables), gets distributed to the federated nodes for executing joins. The partial answers are passed back to the geodistributed analytics where they get aggregated. Event stream analytics can be performed on the data collected by *EdgeAnalyticsBoxes*, which are specifically designed to enable edge analytics or can be performed in any standard hardware with certain processing and storage capabilities.

When implementing edge analytics use cases, it is often required to set up distributed execution of EP systems in a geographically distributed area. Some work has been conducted to investigate collaborations between multiple streaming systems such as CLASP (collaborating, autonomous stream processing systems) [33]. Such systems operate above desperate data stream processing systems and allow them to cooperate. Chandramouli et al. [42] developed a system called *Race* to support a wide variety of distributed feed-following Cloud Edge applications. Race uses a data stream management system (DSMS) for distributedly running different segments of the application execution sequence either on edge devices or in the cloud [42]. Examples of such feed-following Cloud Edge applications include Twitter-like applications (e.g., NanoTwitter on Android phones), context-aware notifiers (e.g., OnX on Android), location-aware coupon services (e.g., GeoQpon on Android and iPhone), and online multiplayer games (e.g., Halo: Reach on Xbox 360), among others.

Lightweight CEP engines have been developed to support edge analytics use cases. These CEP engines essentially have low resource consumption footprints. One example system is JetStream

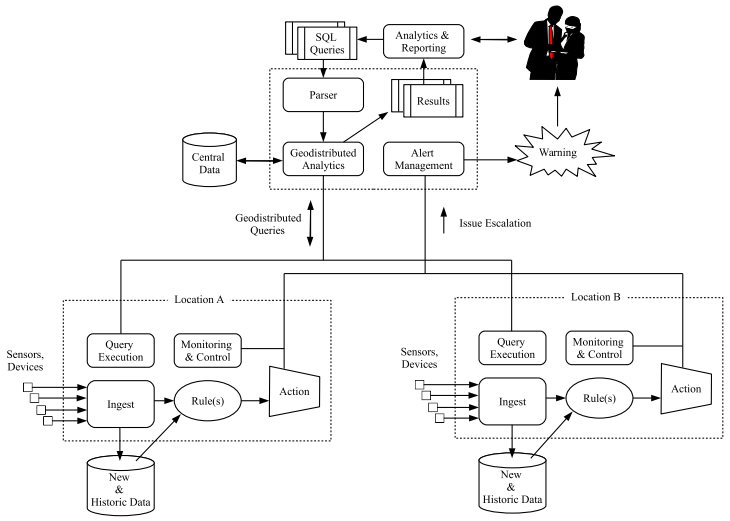

Fig. 3. ParStream's distributed architecture for IoT. The federated mode of operation of ParStream is not significantly different from its single cluster operation.

developed by Santos et al. [163]. JetStream implements a set of context-aware strategies for optimizing batch-based streaming. Therefore, JetStream has the ability to self-adapt to changing streaming conditions.

Edge analytics requires installation of EP hardware near the event sources. This brings the requirements of robust operations in extreme and rugged operating conditions. Furthermore, the edge devices may not be capable of conducting heavy data analysis due to low processing power and/or a low energy envelope. For example, smart meters installed in homes are incapable of handling heavy analysis. Smart meter data could be sent to a gateway that conducts data filtering and sends filtered data to the servers for further processing [156]. How to update the queries running on such hardware is another challenging issue. Most of the current solutions are implemented via dynamic firmware updates.

5.1.2 Context-Aware Event Processing. Context has become a significant abstraction for modeling the EP [66]. Event context and the context awareness are two very important factors in IoT applications. Three key uses of the context in EP applications can be summarized as temporal context, spatial/segment context, and state-oriented context. In temporal context, the stream is divided into windows and the operations are defined in terms of the processing done on the events stored in the window. Spatial/segment-oriented context allows for assigning related events to different context partitions. The EP agent can be active in some contexts and inactive in others, which is called *state-oriented context.* Akbar et al. [12] described a context-aware method to extract and analyze high-level knowledge from data streams. The proposed approach has been implemented on μCEP, which is a lightweight CEP that runs on embedded devices. Similarly, Wang and Cao [195] described a high-performance context-aware CEP architecture and method for the IoT. They modeled the context as fuzzy ontology that supports linguistic variables and uncertainty in event queries.

5.2 Event Processing on Top of Text Data Streams

CEP over text data streams has become a hot topic due to the increased use of online social networking sites, news sites, and e-commerce, among others. The large body of text data stream

processing can be further categorized as stream-based classification, forecasting, and anomaly detection.

Anomaly detection finds patterns in data that deviate from intended characteristics [40]. Spam detection on data streams is one special case of anomaly detection. Twitter has been exploited by spammers to spread malicious messages across the Internet. There have been several notable works conducted on spam detection on Twitter data streams [139]. Online click stream analysis is another subdomain where vendors learn insights about their customer behavior and preferences via processing user clicks [130]. Benhardus and Kalita [27] outlined methodologies of detecting trending topics from streaming Twitter data.

S^3-TM is a stream processing application for streaming short text matching [25]. The work on S^3-TM has shown that text matching can be parallelized by using a partitioning of words over matchers. They have shown that a co-occurrence relationship between words can make word partitioning–based routing a scalable and effective solution that provides more than 2.5 times higher throughput compared to a baseline approach.

5.3 Event Processing on Top of Video Data Streams

CEP over video streams is an important research direction due to the prevalence of video streams captured from surveillance cameras located on stationary platforms [44], mobile ground vehicles, unmanned air vehicles (UAVs), among others [134]. This is very powerful given that cameras are one of the most ubiquitous sensing devices available. Several moving object detection systems have been presented that conduct online analytics on video data streams. Medioni et al. [134] presented a technique for analyzing the behavior of the object movement in a scene. Video streams that are produced by a UAV are processed to detect and track object movement information. Certain research conducts event-based video indexing based on semantical contents. Such work considers multimodal information streams [23]. Zhang and Chang [202] modeled stream processing as a set of task templates whereby each template can be independently instantiated to multiple video streams.

Often, video processing is computationally intensive, and field-programmable gate arrays (FPGAs) have been successfully utilized for accelerating video data stream processing scenarios. Lysakov and Shadrin [126] described an HDG hardware solution for the processing and transmission of TV images, running computation-intensive algorithms, mathematical modeling of numerical methods, and automating physical experiments.

5.4 Analytics on Top of Graph Data Streams

In the past decade, there has been significant interest in developing algorithms for processing massive graphs in the data stream model [133]. Interesting real-world graphs have grown in size from millions to even billions of vertices and edges. For example, in the case of Twitter's information graph, each tweet or search query can be thought of as an edge or a collection of edges in an appropriate graph [62]. In a streaming graph, new edges representing online interactions happening in a network are received by a system as a time-ordered sequence of events. These events need to be added into a much larger graph that represents all historical interactions. A stream of Twitter posts is one example [8]. The area of advancements made on graph data stream processing can be divided into three main areas: novel graph streaming algorithm implementations; streaming graph partitioning, which deals with dividing graph data streams into manageable chunks; and streaming graph processing on linked data (e.g., RDF data stream processing).

Several graph algorithms have been implemented following the data stream processing model. Most of them are streaming implementations of their batched counterparts. One such example is the streaming implementations of the PageRank algorithm. Sarma et al. [52] presented a technique for calculating the PageRank of a large graph. Their approach is based on estimating

the conductance of the graph. Although a standard PageRank algorithm implementation takes $O(n)$ space and $O(M)$ passes, their approach makes an approximation of the PageRank scores in $\tilde{O}(nM^{-\frac{1}{4}})$ space and $\tilde{O}(M^{\frac{3}{4}})$ passes. K-Core decomposition is another important graph algorithm, of which the streaming version was presented by Saríyüce et al. [164]. Pavan et al. [149] presented a technique for triangle counting on a graph stream. Guha et al. [87] presented linear sketches for estimating vertex connectivity and constructing hyper graph sparsifiers. A key feature common across these algorithms is their incremental nature. These algorithms are designed to update their output when a new edge is inserted or removed without needing to traverse the complete graph.

Streaming graph partitioning has become another focus in the graph stream processing community. Stanton and Kliot [180] proposed natural, simple heuristics for streaming graph partitioning and compared their performance to hashing and METIS. They showed that their heuristics provide significant improvement over the best, obtaining an average gain of 76%.

The third important area is linked data stream (LSD) processing. LSD is an extended version of the RDF data model for social network and sensor applications with stream processing functionality [120]. LSD is usually represented as an extension of RDF, which is the most popular standard for linked data representation. However, in contrast to the relational storage model used in DSMSs, to enable efficient processing of RDF data using a relational model, the data need to be heavily replicated. Since replicating a fast-changing RDF stream is prohibitive, the general data stream management architectures cannot be directly used for storage and processing of LSD. In terms of the query operators, a distinguishing characteristic of LSD is the primitive operation on the RDF stream, and the instantaneous RDF dataset is pattern matching that is extended from the triple pattern of SPARQL semantics. There are several notable query languages for LSD processing. One of them is Streaming SPARQL, which extends SPARQL 1.0 grammar. Another significant challenge in LSD is query optimization. It is not straightforward to employ any general optimization techniques for continuous queries over the triple-based data model.

6 EVENT PROCESSING OPEN RESEARCH TOPICS

Different aspects of EP system operation, such as event query languages, accuracy, performance tuning, HA, and FT, as well as novel topics like EP on nonconventional processors and streaming ML, have been addressed more or less by different EP system projects in recent times. In this section, we categorize such work as open research topics and describe recent work conducted in such areas.

6.1 Event Processing Query Languages

Multiple important developments have been made on domain-specific languages for data stream processing. One of the notable advancements is IBM's SPL for InfoSphere Streams [98]. SPL made three key contributions to the area of event query languages by distinguishing itself as an approach for interfacing with nonstreaming languages (e.g., C++ and Java), provides modularity for large streaming applications, and distinguishes itself by its strong type system and operator models.

Multiple research directions of event query languages can be found in recent EP literature. Event query calculus is one such area of high importance. A calculus is a formal system that mathematically defines the behavior of the essential features of a domain, such as ESP [175]. Initiatives such as Brooklet calculus are natural abstractions for streaming languages [174]. The use of a calculus enables formal proofs showing that optimizations made on EP applications are safe and the front-ends realize the semantics of their source languages properly. Another EP calculus related to video stream processing was described by Skarlatidis et al. [172]. TESLA is a rule-based complex event specification language [46]. Each TESLA rule considers incoming data items as notifications of events and defines how complex events are specified from simple ones.

Another important aspect of query languages is the level of expressiveness offered by the query language to describe complex event scenarios. The use of rules and the use of SQL-like query languages are two approaches being followed. Rules enable developers to express application logic in an EP system. Writing rules is a complex and error-prone task that needs to have checking of the correctness of rules with respect to the desired application behavior [50]. As a solution, Cugola et al. [50] provided a methodology and tool for checking a rule set against application-specific properties that have the potential for scaling to large sets of mutually interacting rules. Similarly Zhang et al. [204] conducted theoretical results on expressive power and computational complexity of pattern query languages in EP.

SQL-like queries let end users express complex requirements in a compact form. Furthermore, through operators, they promote the reuse of common logic. Moreover, they enable query planning and automatic optimizations. The use of SQL-like queries has received wide adoption, and all key open source EP systems, such as Apache Strom, Apache Spark Streaming, Apache Fink, WSO2 CEP, and SQLStream, support SQL-like queries.

6.2 Visual Interfaces

Visual interfaces play a key role of uplifting the usability features of modern EP systems. EP application developers need to be isolated from the complexities of hardcore programming while empowering them with the ability of expressing EP scenarios in a simple and intuitive manner. Such visual interfaces especially enable nonprogrammers to write queries easily.

Many real-world products (from IBM, TIBCO, Oracle, Evam, etc.) employ an eclipse-based studio for query composition and results visualization. For example, TIBCO's StreamBase Studio is an Eclipse-based integrated development environment (IDE) using a graphical event-flow language called *StreamSQL EventFlow*, which allows development of real-time applications [187]. IBM InfoSphere Streams Studio is an Eclipse-based developer and administrative workbench [24].

There are two main recent developments in terms of making visualizations of data stream processing: the use of spreadsheets and the use of notebooks. An example of the use of spreadsheets for stream analytics is the work done by Hirzel et al. [99]. They argued that spreadsheets are well suited for programming operators in streaming applications due to several common characteristics found in both spreadsheet and stream programming systems. These include a 2D view of the data, a reactive model of computation, and stateless computation present in both approaches. Their approach proposed a reactive programming model for stream processing that made a uniform treatment of stateless and stateful cells. They also described formal semantics for their core language via a new spreadsheet calculus. They developed a stream processing programming platform called *ACTIVESHEETS* based on Microsoft Excel spreadsheets (Figure 4(a)). ACTIVESHEETS is based on a client-server architecture where the server publishes streams and the client (i.e., the spreadsheet) allows the user to subscribe to live streams and run operations. The client also allows for sharing the results with other users and for persisting the computation on the server beyond the life of the spreadsheet.

Data analytics notebooks have gained huge popularity as a means of conducting interactive data analysis. However, there are very few works being done on the use of notebooks for stream processing. Tempe is a Web application providing an integrated, collaborative environment for both real-time and offline temporal data analysis. Tempe uses Apache Trill as its back-end data processor. Tempe consists of three major components, as shown in Figure 4(b). First, it has a Web client that implements the user experience. Second, it has a notebook service that stores user-generated content and multiplexes requests from multiple clients. Third, it has a scripting service that provides the computing environment for the user's script.

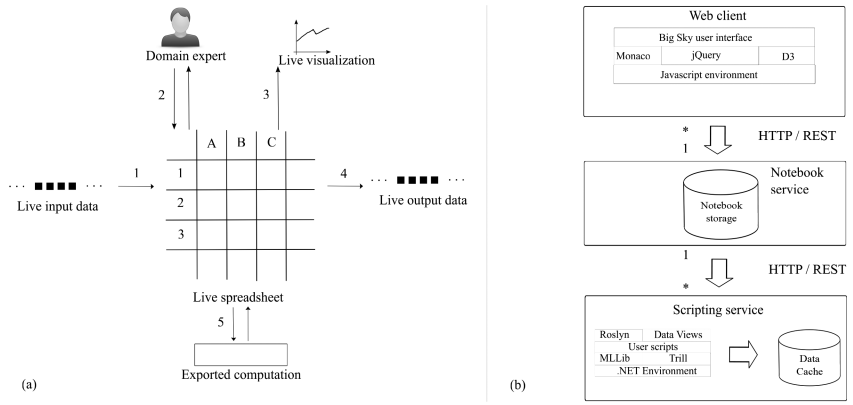

Fig. 4. Different approaches for visualizing data streams: ACTIVESHEETS (a) and Tempe (b).

6.3 Event Processing With Uncertain Data

Uncertain data streams are data streams that comprise incomplete and imprecise data [190]. Imprecise events can be caused by factors such as the deficiencies of the measuring accuracy, signal disturbance, and privacy protection. Feeding such imprecise data to streaming systems produces unexpected results. Hence, this section discusses handling the cases of imprecise data. There are two main types of uncertainty: value uncertainty and existential uncertainty. An event has value uncertainty when its value is represented either by a probability density function (PDF) or by discrete samples. However, an event has existential uncertainty when the sum of the existential probabilities of its possible values is less than 1.

In addressing value uncertainty, cleansing the data is not feasible due to time constraints. Such uncertainties are usually treated with probabilities. Therefore, it has been argued that EP engines that could deal with such probabilities be developed [196]. Mao and Tan [128] proposed a model of an uncertain CEP system for real-time monitoring in the product manufacturing process. They defined probabilistic event models and proposed a probabilistic event detection algorithm based on rNFA and its optimization plan by event filtering.

In addressing existential uncertainty, Dallachiesa et al. [51] have studied the challenges arising from existential uncertainty. They extended the semantics of the sliding window to define the novel concept of uncertain sliding windows. They provide both exact and approximate algorithms for managing windows under existential uncertainty.

A few EP systems are being developed that target uncertain EP. CLARO is a system that supports stream processing for uncertain data [190]. CLARO has a unique data model, called a *mixed-type model*, where continuous uncertain attributes follow Gaussian mixture distributions. Furthermore, it provides formal semantics, query planning for complex queries, and evaluation techniques for relational operators.

An area that needs further attention is the implementation of stream processing algorithms such as heavy hitters, quantiles, and frequent item sets mining over sliding windows. Next, we discuss a special case of uncertain EP called *out-of-order EP*.

6.4 Out-of-Order Event Processing

Out-of-order event arrival is present in general data stream processing applications. Out-of-order events in a stream are produced for multiple reasons, such as operator parallelization, network latency, and merging of asynchronous streams [108]. Orderless EP may result in a system failure. Handling the disorder consists of a trade-off between result latency and result accuracy. There are

four main techniques for disorder handling: buffer based, punctuation based, speculation based, and approximation based.

Buffer-based techniques. Buffer-based techniques sort tuples from the input stream using a buffer before presenting them to the query operator. K-Slack [144] and AQ-K-Slack [108] are two example techniques for buffer-based disorder handling. Buffer-based techniques tolerate a specified amount of "slack" that is basically bound on the degree to which input could be unordered [115]. Buffer-based mechanisms basically delay the processing for a specified slack period (e.g., T). Incoming data is buffered for a T time period, and the reordering of any out-of-order events that arrive within that period is conducted before any intelligent processing on the data has been conducted.

Punctuation-based techniques. Punctuation-based techniques [179] depend on distinct tuples called *punctuations*. These tuples orchestrate returning results pertaining to windows. Hence, unlike buffer-based techniques, out-of-order events can be directly processed by the query.

Speculation-based techniques. Speculation-based techniques operate in the manner that no out-of-order events appear in the stream. When a window gets closed, it produces the results immediately. Already-produced results that get influenced by a late arrived event e are thrown away. After this invalidation, the results are revised by considering the newly arrived e.

Approximation-based techniques. Approximation-based techniques [45] make summaries of the data using some data structure, such as q-digests and histograms. They create rough results using the summary information. Table 5 provides a summary of the content present in this section.

A main disadvantage of buffer-based disorder handling approaches is that sorting and buffering delay the execution of incoming events. This increases the latency of the query results. Punctuation-based disorder handling shares the same issue as buffer-based approaches. Until all late arrivals have appeared on the window, the punctuation that signals reception of all delayed arrivals cannot be created. Speculation-based disorder handling will consume the CPU completely and will create large latency in results with streams having many out-of-order events. Different from speculation-based techniques, approximation-based disorder handling ensures that the delayed events are considered in succeeding aggregate calculations. For queries with small windows, approximation-based disorder handling leads to a considerable number of results with noticeable errors. Overall, buffer-based and punctuation-based approaches are conservative approaches, whereas speculation-based and approximation-based approaches are aggressive. The former two techniques delay their processing to gather delayed events so that they can produce results with higher accuracies. The latter two approaches presume that there is no disorder in the tuples, continue processing as they are, and output the results immediately. They reconcile once a disorder has been detected.

All aforementioned techniques assume that events have precise timestamps so that the order between late events and in-order events is clear. However, there are works that deal with imprecise timestamps and require enumerating all possible ordering among the events [203].

6.5 Event Processing Benchmarking and Performance Characterization

EP system performance is a critical QoS aspect worth investigating. EP benchmarking itself is a vast area with a significant amount of literature. However, EP benchmarking still has several issues to be addressed. There are few widely agreed upon benchmarks for EP. Most of the workload characterization and performance studies have been conducted on microbenchmarks. In this section, we first describe the benchmark construction efforts made on EP. Next, we describe performance characterizations (i.e., workload characterizations) made on EP systems.

One of the earliest and established benchmarks is Linear Road (LR) [22]. It simulates a highway toll system and has been implemented on multiple different CEP systems. Although LR was introduced around 2004, it has been widely implemented in multiple CEP systems (e.g., Botan et al.

Table 5. Summary of Out-of-Order Event Processing Techniques

Approach	Technique	Reference(s)	Characteristics	Limitations
K-Slack	Buffer based	Mutschler and Philippsen [144]	These techniques buffer and reorder events before they are processed by event detectors.	Conservative and overly large K-values result in large buffers. A single fixed a priori K does not work for distributed, hierarchical event detectors.
AQK-Slack	Buffer based	Ji et al. [108]	These techniques dynamically adjust the size of the input buffer to minimize result latency.	These are limited to sliding window aggregates.
Tirtha-pura et al.	Approximation based	Cormode et al. [45] and Tirthapura et al. [188]	These techniques aggregate over polynomial and general decay functions.	They produces a significant amount of errors for queries with small windows.
Brito et al.	Speculation based	Brito et al. [35] and Liu et al. [123]	These techniques are vigorous. They presume that no disorder is present and create results at the same time the window closes. They assume in-order arrival of tuples and produce the results of a window immediately when it is closed.	Several iterations of revisions are conducted to arrive at the final revision. This will consume a lot of CPU and introduce large latency in the results.
Krishnam-urthy et al.	Punctuation based	Krishnamurthy et al. [115]	These techniques depend on punctuations within data streams. Punctuations explicitly trigger the operator to send results.	The latency problem of buffer-based techniques still exists with this approach.

Table 6. Summary of Event Processing Benchmarks

Benchmark	Reference(s)	Characteristics	Limitations
Linear Road	Arasu et al. [22]	Application	Complex to implement, difficult to parallelize
FINCoS	Mendes et al. [137]	A set of benchmarking tools for load generation and performance measurement	Limited scalability
Email Processor	Nabi et al. [145] and Ravindra and Dayarathna [157]	Application	Does not include EP operations such as windows
CityBench	Ali et al. [14]	RDF stream processing benchmark	Limited scalability

[32]). Among these implementations, the LR implementation developed for SCSQ by Zietler and Rische [201] is of importance.

Another notable work on EP benchmarking was conducted by Mendes et al. [137] called *BiCEP suite*. This framework consists of several benchmark specifications (e.g., Pairs), as well as a workload generator framework called *FINCoS*. Nabi et al. [145] implemented a benchmark based on the Enron email corpus. CEPSim is a simulator for cloud-based EP systems that can be used to study the performance and scalability of EP queries [94]. CEPSim can be used to model different types of clouds and to simulate execution of user-defined queries on them.

Efforts have been made in developing performance models for stream processing systems as a means of formalizing the process of characterizing and predicting the performance of EP systems. In one such work, Bedini et al. [26] presented a set of models that formalize the performance characteristics of a practical stream processing system that follows the actor model. They model the characteristics of the dataflow cost, the data processing cost, and the system management costs at a fine granularity within the different steps of executing a distributed stream processing job.

Chauhan et al. [43] conducted an empirical evaluation of the Yahoo! S4 stream processing platform with a word processor EP application. The application converts each and every word of a text corpus to numbers and conducts simple filtering and counting to find numbers that can be divided either by 3 or 11. Similarly, Xu et al. [198] conducted a performance study of the T-Storm stream processing system with three benchmarks: Throughput Test, Word Count, and Log Stream Processing. Nabir et al. [145] conducted a detailed study of Storm's performance. Several works, such as those of Dayarathna and Suzumura [58] and Mendes et al. [136], conducted workload characterizations across different EP systems. Factors affecting a virtualized EP system were analyzed by Ku et al. [117].

There have been recent studies conducted that consider other novel dimensions of performance of stream processing software, such as those of Dayarathna et al. [55] and Sun et al. [182]. However, they characterize the energy consumption of stream processing systems. Furthermore, certain CEP benchmarking works have considered scenarios with out-of-order events [84].

Multiple studies have been conducted to assess the performance characteristics of similar CEP+Storm systems, such as SQL Stream Blaze [176], Amazon Kinesis [28], and WSO2 CEP [157].

The DEBS Grand Challenge is a notable initiative made by the EP community to characterize and improve the performance of EP systems. The DEBS Grand Challenge was introduced to the DEBS conference series in 2011 by Opher Etzion. Each year, the competition introduces a unique real-world application scenario of EP and ranks the solutions mainly based on performance (e.g., throughput and latency) of the solution. Since its inception, the DEBS Grand Challenge has attracted significant attention from the EP community. In the most recent version of the challenge [106] (DEBS'15), 14 solutions were accepted by the organizers.

There have been recent work conducted on developing benchmarks for specific use cases such as RDF stream processing. CityBench is one such example that was developed for evaluating RDF SPEs within smart city applications and with smart city data [14]. The benchmark consists of real-time IoT data streams generated by multiple sensors deployed within the city of Aarhus, Denmark. Two RDF SPEs were evaluated using CityBench to evaluate their capabilities and limitations to operate as RDF SPEs in smart city applications.

6.6 Event Processing Provenance

Provenance is a detailed record of how an output data element is produced by a system, including detail about the inputs it received. It is used to describe the incoming data and the intermediary state that made it to produce such output [140]. EP provenance is related to a new class of

applications that require diagnosing capabilities, human observation, and assurance [80, 81]. EP provenance helps human supervisors diagnose how certain events were triggered in the system and react accordingly.

Glavic et al. [81] described an approach for stream processing provenance that uses operator instrumentation. Their approach sends detailed provenance information via some operators by modifying operators to generate and transfer such information through the query network. They studied how to reduce the effect of provenance computation from query execution.

A high stream event rate and low latency are the two key performance requirements with most EP applications. Hence, when enabling background processes such as provenance, it is important that they do not degrade the performance numbers. Century is an example online medical analytics infrastructure with support for provenance with such performance requirements [140]. Their approach was named *composite modeling with intermediate replay* (CMIR), which created a solution for the problem of stream persistence. Their approach defined dependency relationships only on a subset of processing elements of the stream processing network and recreated data elements of streams through data replay. Through this hybrid provenance technique, they avoided the requirement of storing external and intermediate streams that otherwise would have to be handled by their systems.

6.7 Data Stream Processing in the Clouds

Cloud computing has transformed many aspects of modern data analysis, including EP [91]. Multiple works have been conducted in conjunction with EP in cloud environments. Amazon Kinesis Streams is an example effort made by the industry [16]. Clouds have their own inherent issues, with key issues being FT, scalability and load balancing, security, and privacy. This section reviews the recent state-of-the art work done on the last two areas specifically. We discuss the details of FT in cloud computing systems in Section 6.8.

Elastic load balancing is essential to operate between multiple different clouds. Kleiminger et al. [113] presented a combined stream processing system that adaptively balances the workload between two stream processors deployed in both a private cloud and a public cloud. They implemented and evaluated their approach with a financial trading scenario. It was shown that the bandwidth between the load balancer and the cloud data center acts as the key factor of load balancing in such scenarios. Cervino et al. [37], however, mentioned that the main factor dominating the performance of elastic load balancing with clouds is the latency introduced by the cloud deployment. Ishii and Suzumura [104] created a method and an architecture that transfers data stream processing to a cloud environment as required by monitoring the data rate of the input data stream. Heinze et al. [93] introduced an elastic data stream processing system that optimizes its utilization under certain latency constraints defined by an SLA. It was developed as an online parameter optimization.

Privacy and security are of utmost importance in the context of cloud computing. Multiple techniques have been exploited for implementing privacy preserving stream data analytics that can be effectively used for securing the streaming data in clouds. Out of the several algorithmic techniques, the randomized method has been used in this regard. The randomization method adds noise to the streaming data to mask the attribute values of the streaming events [9]. The noise is a significant amount, which makes the individual event information not recoverable, but aggregate distributions can be recovered through algorithmic techniques. Chan et al. [39] described a mechanism for collecting privacy-sensitive data from users and calculate aggregate statistics periodically. They introduced a new technique to achieve FT while incurring a very small penalty of communication overhead and estimation error.

6.8 High Availability and Fault Tolerance

HA and FT are closely related yet different, and they are often confused. In this section, we first differentiate between these two terms. Next, we describe how HA has been implemented in EP systems and then move onto describing different approaches that are followed to implement FT.

A system is highly available if it operates with tolerable downtime [112]. It is impossible for a system to have zero down time. HA is critical for operating a CEP system smoothly. HA enables system to be accessible 24/7. However, HA mechanisms do not guarantee the fault-free operation of the application. Fault-tolerant systems are capable of operating even after a system failure occurs [112]. FT in an EP system is much more complex to implement compared to implementing HA.

We achieve HA though replicating the processing. There are two main approaches for implementing HA in EP systems: active standby (AS) and passive standby (PS) [86, 103]. In AS, two or more copies of the same job are run independently on different compute nodes. In PS, a primary copy periodically checkpoints its state to another machine and uses that copy for recovery during failures. Zhang et al. [205] described a hybrid HA method that combines the benefits of both AS and PS approaches. The system operates in PS mode during normal conditions and quickly switches to the AS mode by activating a predeployed secondary copy that was in suspension. Once the primary copy becomes responsive again, the system rolls back to the PS mode. This way, the system faces small overhead most of the time while providing fast recovery during failures and unavailability.

Fault-tolerant operations of an EP system are defined in terms of delivery guarantees found in messaging systems: At Least Once, At Most Once, and Exactly Once. EP software such as Storm, MillWheel [13], Spark Streaming [200], Apache Flink, and S-Store [135] have used these guarantees [54]. With *Exactly Once* even in the presence of failures, the program's state will eventually reflect every record from the data stream exactly once. There are less strict versions of the delivery guarantee, called *At Least Once* and *At Most Once*. At Least Once may produce at least once guaranteed delivery of an event but may produce more than one copy of the event in some cases. At Most Once may process each tuple at most once. This technique intentionally drops the events to maintain the freshness of the results. Exactly Once is followed in microbatching systems such as Spark Streaming, Trident, transactional updates systems such as Google Cloud Dataflow, and distributed snapshots systems such as Flink. Storm, however, follows an At Least Once guarantee that results in low throughput. A detailed comparison of these FT models is available in Tzoumas et al. [192]. Flink is a good example of a project that has Exactly Once, where it creates consistent checkpoints of an EP graph using variation of global snapshot algorithm. At Least Once can be implemented using a persistent message queue and client acknowledgements before removing the message from the queue.

Support for ACID transactions in streaming systems is another recently discussed topic in the CEP community that is important for fault-tolerant operation of EP systems. Several EP systems have appeared to address the issue of introducing transactional guarantees to the data stream processing. S-Store is an example of a system [38] in this category. S-Store builds on H-Store, which is a high-performance, in-memory, distributed OLTP system designed for shared-nothing clusters. All data access in S-Store is SQL based and fully transactional. It addresses the shortcomings of legacy stream processing systems' ability for state management. Google Cloud Dataflow [116] is another example of an EP that follows the transactional updates model [192]. In such a model, each intermediate record that passes through an operator (together with the derived records and state updates) creates a commit record that is atomically appended to a transactional log or inserted to a database. When a failure is encountered, part of the database log is replayed to consistently restore the state; replay of the lost records is also carried out. However, the transactional updates

model depends on the ability to write frequently to a distributed fault-tolerant store with high throughput, which limits its use outside Google's infrastructure.

Fault-tolerant operation of EP jobs in clouds is of critical importance for wide adoption of cloud computing for big data streaming processing. System failures are inevitable in cloud computing systems with thousands of servers. Powerful abstractions have been designed to cope with the failures that are present in cloud computing systems, such as the work done on TimeStream [153]. TimeStream has a mechanism that tracks fine-grain data dependencies between the output and input streams to enable efficient recomputation-based failure recovery that achieves strong Exactly Once semantics.

EP systems often have to balance the two QoS parameters of latency and throughput. Lohrmann et al. [124] described an approach that is based on dynamic task chaining and adaptive output buffer sizing to achieve latency constraints defined by users. Taking video streaming as the use case, they implemented and evaluated their approach. Their approach improved the workflow latency by at least 15 times with regard to maintaining the necessary throughput value [124].

6.9 System Scalability and Autotuning

Scalability is the system's ability to handle increasing amounts of workload gracefully, or its capability of expanding effortlessly and transparently to support such growth [67]. This can be done through large machines (vertical scaling) or through the use of multiple machines (horizontal scaling).

In the era of multicore computing, EP software systems need to utilize all available cores of a computer system in an efficient manner. EP optimizations can be categorized under two main areas; black-box optimizationsdistribution, parallelism, scheduling, load balancing, load shedding, etc.) and white-box optimizations (implementation selection, implementation optimization, pattern rewriting, etc.) [67]. In EP systems, parallelism can be achieved through one of three techniques: data parallelism, task parallelism, and pipeline parallelism [165].

DSCP scalability is a widely studied area due to the appearance of several open source DSCPs including Apache Storm and Apache Spark Streaming. New mechanisms created for scalability improvement of DSCPs focus significantly on scheduling algorithm used in such systems. The works by Aniello et al. [18] and T-Storm [198] are examples of such recent efforts. StreamCloud is another system in this category [88]. One of the main challenges of an elastic system is to find the right point in time to scale in or scale out because of a constantly changing workload and system characteristics. Heinze et al. [92] implemented three autoscaling strategies: global thresholds, local thresholds, and reinforcement learning.

Autoparallelization of data stream processing systems is another area that is widely being studied. Several autoparallelization works have been conducted in the context of IBM InfoSphere Streams (i.e., System S). Dayarathna et al. [56, 57] described dataflow graph transformation as a technique for autotuning a streaming application run in System S. Schneider et al. [165] and Gedik et al. [77] addressed the issue of autotuning with the presence of stateful operators.

Another technique for the performance optimization of CEP systems is pattern rewriting. Rabinovich et al. [154] showed that pattern rewriting can introduce more than 10 times improved performance between the original pattern and its rewritten counterpart. A closely related technique for pattern rewriting is operator reordering. It includes techniques such as hoisting, sinking, rotation, and pushdown [100].

Operator placement in a distributed EP system determines its scalability to a greater extent. Cugola and Margara [49] conducted a comparison of various deployment techniques that a CEP middleware could follow based on how the processing load is distributed over different processors and how processors interact to produce the required results.

Table 7. Comparison of Event Processing Scaling Approaches

Approach	Reference(s)	Advantages	Limitations	Comments
Fission (i.e., Paral-lelization)	Dayarathna and Suzumura [57] and Gedik [77]	Leverages multicores	Difficult to partition a shared state	Dynamic determination of the level of parallelism increases the performance portability.
Fusion	Dayarathna and Suzumura [57], Gedik [77], and Khandekar et al. [111]	Reduced complexity of accessing shared state	Cannot leverage multicores	This can be done either following top-down or bottom-up approaches.
Pipelining	Gordon et al. [83]	Leverages multicores	Adds additional path to the flow graph per each pipeline stage created	Pipelining may enable other optimizations, such as operator reordering.
Rewriting	Rabinovich et al. [154] and Schultz-Møller [167]	Improved performance on the same hardware environment	Application specific	This is suited for applications that involve complex pattern detection.
Operator reorder-ing	Pollner et al. [152]	Improved performance on the same hardware environment	Very restrictive preconditions for safe operator graph changes	This is fruitful if it moves selective operators before costly operators.

Due to the highly dynamic nature of the input data stream, identifying a single plan results in a suboptimal query plan. The multiroute approach, which is a new stream processing technique, has appeared to solve this issue [36]. Cao et al. [36] described one of the first practical multiroute optimizers, called the *correlation-aware multiroute stream query optimizer* (CMR), which solves the preceding issue. In this article, we describe the most commonly encountered scaling techniques. It can be observed that system scalability experiments conducted until now are focused on full system performance. However, system performance characteristics in special scenarios, such as performance behavior during system failures, data partitioning, and windowing, need to be addressed in detail. A summary of these techniques is listed in Table 7.

6.10 Approximate Complex Event Processing

The development of approximation algorithms for streaming data has become very important due to the prominence of large volumes of data streams. Approximation algorithms can produce significant performance improvements (e.g., latency) for such big data streaming applications.

There has been extensive research conducted on developing compact data structures for summarizing large data streams. One of the families of such data structures is called *sketches*.

Dimitropoulos et al. [60] addressed the problem of sketch data structures getting saturated, which results in a potentially large error on estimated key counts. Their solution detected transient keys and periodically removed their hashed values from the sketch. Both techniques that they introduced either slowed down the saturation process of a sketch or completely prevented a sketch from saturating. Papapetrou et al. [148] addressed the problem of complex query answering over distributed, high-dimensional data streams in the sliding window model.

Rusu and Dobra [161] conducted an empirical comparison between various sketching techniques proposed in the literature from a statistical point of view with the goal of identifying the actual behavior and producing tighter confidence bounds. Garofalakis et al. [76] proposed a novel algorithmic technique for efficiently tracking sketch-based approximations for a broad class of complex aggregate queries over massive, distributed data streams. The tracking protocols that they developed are based on a combination of the geometric method of Sharfman et al. [168] for monitoring general threshold conditions over data streams and Alon, Matias, Szegedy (AMS) sketch estimators for querying massive streaming data. By exploiting properties of AMS sketches, their algorithm can perform highly efficient geometric monitoring in a much lower-dimensional space. They also proposed novel geometric algorithms for tracking medians computed over AMS sketches of the streams for different types of distributed stream queries of high practical interest. Their technique has shown consistent gain up to 35% in terms of full communication cost compared to the state of the art.

Aggarwal and Yu [10] discussed the use of sketching techniques for historical diagnosis of sensor networks. They specifically discussed methods that are accurate enough to reconstruct any portion of the historical sensor signal with a relatively small amount of prestored data. The storage requirements for big data streams is significant over time, and a compressed representation is helpful in such space-constrained scenarios.

6.11 CEP on Nonconventional Processors (GPUs and FPGAs)

Advancement in the use of nonconventional high-performance device technologies such as GPUs and FPGAs provide an opportunity to build very fast event-based systems. High-performance, massively parallel accelerators such as GPUs and FPGAs have been recently deployed in CEP applications. There are few attempts made on implementing complex event processors on top of such devices.

Glacier is a component library and a compositional compiler that transforms continuous queries into logic circuits by creating library components on an operator-level basis [141]. The goal of developing Glacier was to create a hybrid data SPE where an optimizer distributes query workloads across a set of CPUs and FPGA chips. Mueller et al. [141] investigated how conventional streaming operators can be mapped onto an FPGA, how they can be combined into queries over data streams, the proper system setup for the FPGA to operate in combination with an external CPU, and the actual performance that can be reached with the resulting system. They found that it is really difficult to generate really high packet rates in lab settings to stress the FPGA. Using an NetBSD-based packet generator, the authors generated a maximum of 1,000,400 packets/s on UDP. They showed that the FPGA can process streams at network speed where the bottleneck exists with the network.

It is difficult to use a multicore processor like Cell for EP, mainly due to the heterogeneous nature of the processing elements. The coprocessor side local memory and the unconventional programming model usually inhibit their widespread adoption [78]. In search of a solution Gedik et al. studied the execution of windowed stream join operators in a scalable manner on multicore processors (specifically targeting Cell processors [78]). Their approach achieves 8.3× higher output rate compared to a naive approach on two 3.2GHz Intel Xeon processors.

Cugola et al. [47] investigated how parallel hardware may speed up CEP. They considered the most common operators offered by the current rule languages (sequences, parameters, aggregates, etc.) and considered different algorithms to process rules built using such operators. They discussed how such algorithms can be implemented on a multicore CPU and on CUDA. Verner et al. [193] designed a framework for hard real-time stateful processing of multiple streams on heterogeneous platforms with multiple CPUs and a single accelerator. Margara and Cugola [129] showed that the use of parallel hardware can provide impressive speedups in content-based matching. They developed the parallel content matching (PCM) algorithm to leverage parallel hardware to perform publish-subscribe content-based matching. Giakoumakis et al. [79] described a novel FPGA-based system to accelerate the classifier building process over data streams, which achieves two orders of magnitude performance over the software-based solutions. In particular, they mapped the very fast decision tree (VFDT) streaming algorithm on a single custom processor in a high-end Convey HC-2ex FPGA platform. Karnagel et al. [110] proposed a variant of a window-based stream join operator that is optimized for processors with heterogeneous execution units. The join operation is performed within windows of the stream and gets scheduled as a task within a queue. Furthermore, they proposed the HELLS-join approach, which uses all heterogeneous devices by segregating parts of the algorithm onto the appropriate device [109]. They have shown that HELLS-join performs better than CPU stream joins, which allows for the use of wider time windows, higher stream frequencies, and more streams to be joined.

Koromilas et al. [114] proposed an adaptive scheduling approach that supports heterogeneous and symmetric hardware tailored for network packet processing applications. They showed that their system is capable of matching the peak throughput of a diverse set of packet processing workloads while dissipating up to 3.5× less energy.

6.12 Streaming Processing With Database Management Systems

EP has its roots database management system (DBMS) research. It is well known fact that the state-of-the-art database technology cannot handle streams efficiently due to their continuous nature. Relational database management systems (RDBMSs) are the main type of database engines currently used for data management use cases. Hence, it has been increasingly important to investigate the use of RDBMSs for data stream processing. However, a few works have been conducted by the NoSQL community on combining streaming, in-memory, and historical data in real-time analytics [119]. In this section, we highlight some of the work done in this area.

Liarou et al. [122] designed a stream engine on top of an existing relational database kernel. This enabled them to reuse its storage/execution engine and its optimizer infrastructure. They worked on maintaining and reusing the generic storage and execution model of the DBMS at the query plan level. Joining with data in an RDBMS is conducted via windows, where every time a window is complete, the result over all tuples in that window is calculated.

The feasibility of using database systems to assist SPEs in processing complex aggregate queries to reduce their evaluation latency has been an area of investigation [107]. Dynamic migration of complex aggregate operations between the SPE and the database engine in response to varying system loads have been investigated by Ji et al. [108].

6.13 Streaming Machine Learning

ML learns an underline function given sample data [1]. Most of the ML algorithms are initially developed targeting batched data analytics. A lot of ML analysis problems involve fitting models to the data as the data is being acquired and have the models update over time. Furthermore, in recent times, the emergence of Web-scale datasets in the Internet companies has resulted in the

Table 8. Comparison of Streaming Machine Learning Projects

Project	Example Real-World Applications	Distri-buted?	Language	Limitations	Comments
SMOA	Web data mining	Yes	Java	Works best for large, fast data	Supported execution engines include Storm, S4, Samza, and Flink.
Jubatus [169]	Measure online real estate service customer preference	Yes	C++	—	It is based on loose model sharing for online distributed learning and prediction.
Mahout	Support call categorization [61]	Yes	Java	Depends on Hadoop	—
VFML [102]	Yes	No	C	Cannot scale across a compute cluster	—
MLlib [138]	Yes	Yes	Scala	—	—

Note: All projects are open source ML.

necessity to conduct ML on streaming data. Several libraries have appeared to solve the problem of Streaming ML.

Scalable Advanced Massive Online Analysis (SAMOA) is a platform for mining big data streams [59]. It consists of a collection of distributed streaming algorithms for general data mining and ML activities such as classification (vertical Hoeffding tree (VHT)), clustering (CluStream), and regression (distributed Adaptive Model Rules (HAMR)). SAMOA consists of a pluggable architecture that allows it to run on several DSCPs.

Stream clustering has been a widely studied algorithm. One of the example variations in this area is clustering large-scale streaming graphs [62]. In this scenario, the updates to the graph are given in the form of a stream of edge or vertex additions or deletions. They used graph reservoir sampling to design a streaming algorithm for clustering vertices in graphs.

Concept drift is a phenomenon related to an online supervised learning scenario where the relation between the input data and the target variable changes over time [74]. The main assumption behind concept drift is that the generating function of the new data is unknown to the learner, which makes the concept drift unpredictable [101]. To solve this issue, one must design a unified classifier that can handle such changes in concepts over time. Bifet and Gavaldà [29] took Hoeffding tree, an incremental decision tree inducer for data streams and used as a basis to build two new methods that can deal with distribution and concept drift.

Most streaming decision models evolve continuously over time. Gama et al. [73] proposed a general framework to evaluate predictive stream learning algorithms. Their experiments showed that the use of forgetting mechanisms is required for fast and efficient change detection. A comparison of some of the notable streaming ML projects is listed in Table 8.

Open Research Topics in CEP	Query languages and visual interfaces						2		1	1	1	4
	Uncertainty								1	1		2
	Nonconventional devices					2		1	1	2	2	1
	Approximate CEP		1	1	1				1	1	1	2
	System scalability and autotuning						1	2	2	4	4	
	HA and FT	1				1	1		1	4	1	4
	Provenance				1					1	1	
	Benchmarking and performance characterization					1	1	1	2	4	5	6
	Out-of-order events				1		2				1	1
	Streaming ML					1			2	1	1	1
	Stream processing in clouds				1			2	2		2	1
CEP Use Cases	Graph data streams				1				3	1	2	1
	Video data streams									1	1	
	Text data streams					1				1	2	1
	IoT use cases			1				1	1	3	4	3
CEP System Architectures	DSCPs						2			1	3	5
	Event processing platforms					1	1			1	2	3
	CEP libraries			1				1		2	1	2
Time (Year)		2005					2010					2015

Fig. 5. Advancements made in EP. The timeline provides a summary of the main topics surveyed in this article.

7 CONCLUSIONS

EP has been a paradigm for data processing for nearly three decades. We have observed significant advancements made in EP over the past 5 to 10 years. This article created a conscious summary of the research by classifying the entire body of research into three categories: EP system architectures, EP use cases, and open research topics. It can be observed that significant work carried out in the EP system architectures has especially focused on the DSCPs. Among the notable EP use cases, EP on the IoT, EP on text data streams, and data stream processing in clouds remain significant areas.

In terms of the open research topics, we have observed significant amounts of work conducted recently on event ordering, system scalability, formalizing and development of EPLs, and the use of heterogeneous devices for EP (Figure 5).

Although most of the modern open source EP systems have been developed in Java, there is a significant trend by the large-scale data analysis community to use Scala as a new language for developing EP systems. These trends are reflected in stream processing environments such as Spark Streaming. Furthermore, most of the current EP systems do not have SQL-like query languages, which pose significant usability issues by novice users of such technologies.

Real-world implementations of streaming ML algorithms remain in their early stages. There have been significant recent efforts made on IoT-related EP. Graph streaming processing is another interesting area that is still in its infancy. We envision promising research advances in the areas of the IoT, graph streams, and Streaming ML in the near future.

A significant amount of work needs to be done with regard to EP benchmarking aspects. Although several EP benchmarks have been created, they do not address the complete range of applications confronted by EP systems. For example, graph stream processing is currently an emerging area, but none of the existing benchmarks address the scenarios of graph stream processing. Furthermore, there is no concession among researchers in EP on a standard benchmark for system performance.

Relatively few works has been carried out in event pattern matching on graph streams [173]. Application complexity and the performance metrics used for such benchmarking are some other areas to be addressed. ML techniques can be used for constructing optimized query plans for EP systems. Some work in this domain was conducted previously in the context of embedded systems [197].

Although Streaming ML has attracted significant attention recently, not many real-world implementations of the streaming ML algorithms are available at this time. Furthermore, such algorithms' performance behaviors remain unknown.

In summary, the preceding analysis indicates that there has been significant growth in EP applications and related technologies over the past 5 to 10 years. We expect similar growth in the EP industry over the next decade, as highlighted by our analysis.

REFERENCES

[1] Norbert M. Seel (Ed.). 2012. Mathematical models. In *Encyclopedia of the Sciences of Learning*. Springer US, 2113–2113.

[2] Research and Markets. 2014. *Complex Event Processing (CEP) Market—Global Forecast to 2019*. Research and Markets.

[3] Dell. 2015. Dell Edge Gateway 5000 Series. Retrieved January 22, 2018, from http://i.dell.com/sites/doccontent/corporate/secure/en/Documents/edge-gateway-specsheet.pdf.

[4] Research and Markets. 2015. *Streaming Analytics Market by Verticals—Worldwide Market Forecast and Analysis (2015-2020)*. Research and Markets.

[5] C. Cabanillas C. Di Ciccio R. Eid-Sabbagh M. Hewelt A. Meyer A. Rogge-Solti A. Baumgrass, R. Breske. 2014. S-Store: Streaming meets transaction processing. arXiv:1503.01143.

[6] Daniel J. Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Cetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, et al. 2005. The design of the borealis stream processing engine. In *Proceedings of the 2nd Biennial Conference on Innovative Data Systems Research (CIDR'05)*, Vol. 5. 277–289.

[7] Raman Adaikkalavan and Sharma Chakravarthy. 2006. SnoopIB: Interval-based event specification and detection for active databases. *Data and Knowledge Engineering* 59, 1, 139–165. DOI: http://dx.doi.org/10.1016/j.datak.2005.07.009

[8] Charu Aggarwal and Karthik Subbian. 2014. Evolutionary network analysis: A survey. *ACM Computing Surveys* 47, 1, Article 10, 36 pages.

[9] Charu C. Aggarwal and Philip S. Yu. 2008. A general survey of privacy-preserving data mining models and algorithms. In *Privacy-Preserving Data Mining*, C. C. Aggarwal and P. S. Yu (Eds.). Advances in Database Systems, Vol. 34. Springer US, 11–52.

[10] C. C. Aggarwal and P. S. Yu. 2015. On historical diagnosis of sensor streams. In *Proceedings of the 2015 IEEE 31st International Conference on Data Engineering (ICDE'15)*. 185–194.

[11] Charu C. Aggarwal. 2013. A survey of stream clustering algorithms. In *Data Clustering: Algorithms and Applications*. CRC Press, Boca Raton, FL, 231–258.

[12] A. Akbar, F. Carrez, K. Moessner, J. Sancho, and J. Rico. 2015. Context-aware stream processing for distributed IoT applications. In *Proceedings of the 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT'15)*. 663–668. DOI: http://dx.doi.org/10.1109/WF-IoT.2015.7389133

[13] Tyler Akidau, Alex Balikov, Kaya Bekiroğlu, Slava Chernyak, Josh Haberman, Reuven Lax, Sam McVeety, Daniel Mills, Paul Nordstrom, and Sam Whittle. 2013. MillWheel: Fault-tolerant stream processing at internet scale. *Proceedings of the VLDB Endowment* 6, 11, 1033–1044.

[14] Muhammad Intizar Ali, Feng Gao, and Alessandra Mileo. 2015. CityBench: A configurable benchmark to evaluate RSP engines using smart city datasets. In *Proceedings of the 14th International Semantic Web Conference (ISWC'15), Part II*. 374–389.

[15] Muhammad Intizar Ali, Naomi Ono, Mahedi Kaysar, Keith Griffin, and Alessandra Mileo. 2015. A semantic processing framework for IoT-enabled communication systems. In *Proceedings of the 14th International Semantic Web Conference (ISWC'15), Part II*. 241–258.

[16] Amazon Web Services Inc. 2016. Amazon Kinesis Data Streams. Retrieved January 22, 2017, from https://aws.amazon.com/kinesis/streams.

[17] Amineh Amini, TehYing Wah, and Hadi Saboohi. 2014. On density-based data streams clustering algorithms: A survey. *Journal of Computer Science and Technology* 29, 1, 116–141.

[18] Leonardo Aniello, Roberto Baldoni, and Leonardo Querzoni. 2013. Adaptive online scheduling in storm. In *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems (DEBS'13)*. ACM, New York, NY, 207–218.

[19] M. Anis Uddin Nasir, G. De Francisci Morales, D. Garcia-Soriano, N. Kourtellis, and M. Serafini. 2015. The power of both choices: Practical load balancing for distributed stream processing engines. In *Proceedings of the 2015 IEEE 31st International Conference on Data Engineering (ICDE'15)*. 137–148.

[20] Apache Software Foundation. 2015. Apache Flink: Scalable Batch and Stream Data Processing. Retrieved January 22, 2018, from https://flink.apache.org/.

[21] Apache Software Foundation. 2015. What Is Samza? Retrieved January 22, 2018, from http://samza.apache.org/.

[22] Arvind Arasu, Mitch Cherniack, Eduardo Galvez, David Maier, Anurag S. Maskey, Esther Ryvkina, Michael Stonebraker, and Richard Tibbetts. 2004. Linear Road: A stream data management benchmark. In *Proceedings of the 30th International Conference on Very Large Data Bases, Vol. 30 (VLDB'04)*. 480–491.

[23] N. Babaguchi, Y. Kawai, and T. Kitahashi. 2002. Event based indexing of broadcasted sports video by intermodal collaboration. *IEEE Transactions on Multimedia* 4, 1, 68–75.

[24] C. Ballard, D. M. Farrell, M. Lee, P. D. Stone, S. Thibault, S. Tucker, and IBM Redbooks. 2010. *IBM InfoSphere Streams Harnessing Data in Motion*. IBM Redbooks.

[25] Fuat Basık, Buğra Gedik, Hakan Ferhatosmanoğlu, and Mert Emin Kalender. 2015. S3-TM: Scalable streaming short text matching. *VLDB Journal* 24, 6, 849–866.

[26] Ivan Bedini, Sherif Sakr, Bart Theeten, Alessandra Sala, and Peter Cogan. 2013. Modeling performance of a parallel streaming engine: Bridging theory and costs. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering (ICPE'13)*. ACM, New York, NY, 173–184.

[27] James Benhardus and Jugal Kalita. 2013. Streaming trend detection in Twitter. *International Journal of Web Based Communities* 9, 1, 122–139.

[28] Rahul Bhartia. 2014. *Amazon Kinesis and Apache Storm: Building a Real-Time Sliding-Window Dashboard Over Streaming Data*. Technical Report. Amazon Web Services.

[29] Albert Bifet and Ricard Gavaldà. 2009. Adaptive learning from evolving data streams. In *Proceedings of the 8th International Symposium on Intelligent Data Analysis: Advances in Intelligent Data Analysis VIII (IDA'09)*. 249–260.

[30] Robin Bloor and Rebecca Jozwiak. 2014. *A Database Platform for the Internet of Things*. White Paper. Available at https://insideanalysis.com/research/white-papers/.

[31] M. Blount, M. R. Ebling, J. M. Eklund, A. G. James, C. McGregor, N. Percival, K. P. Smith, and D. Sow. 2010. Real-time analysis for intensive care: Development and deployment of the artemis analytic system. *IEEE Engineering in Medicine and Biology Magazine* 29, 2, 110–118.

[32] I. Botan, Y. Cho, R. Derakhshan, N. Dindar, A. Gupta, L. Haas, K. Kim, et al. 2010. A demonstration of the MaxStream federated stream processing system. In *Proceedings of the 2010 IEEE 26th International Conference on Data Engineering (ICDE'10)*. 1093–1096.

[33] Michael Branson, Fred Douglis, Brad Fawcett, Zhen Liu, Anton Riabov, and Fan Ye. 2007. CLASP: Collaborating, autonomous stream processing systems. In *Middleware 2007*. Lecture Notes in Computer Science, Vol. 4834. Springer, 348–367.

[34] Lars Brenna, Alan Demers, Johannes Gehrke, Mingsheng Hong, Joel Ossher, Biswanath Panda, Mirek Riedewald, Mohit Thatte, and Walker White. 2007. Cayuga: A high-performance event processing engine. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data (SIGMOD'07)*. ACM, New York, NY, 1100–1102.

[35] Andrey Brito, Christof Fetzer, Heiko Sturzrehm, and Pascal Felber. 2008. Speculative out-of-order event processing with software transaction memory. In *Proceedings of the 2nd International Conference on Distributed Event-Based Systems (DEBS'08)*. ACM, New York, NY, 265–275.

[36] Lei Cao and Elke A. Rundensteiner. 2013. High performance stream query processing with correlation-aware partitioning. *Proceedings of the VLDB Endowment* 7, 4, 265–276.

[37] J. Cervino, E. Kalyvianaki, J. Salvachua, and P. Pietzuch. 2012. Adaptive provisioning of stream processing systems in the cloud. In *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering Workshops (ICDEW'12)*. 295–301.

[38] Ugur Cetintemel, Jiang Du, Tim Kraska, Samuel Madden, David Maier, John Meehan, Andrew Pavlo, et al. 2014. S-Store: A streaming NewSQL system for big velocity applications. *Proceedings of the VLDB Endowment* 7, 13, 1633–1636.

[39] T.-H. Hubert Chan, Elaine Shi, and Dawn Song. 2012. Privacy-preserving stream aggregation with fault tolerance. In *Financial Cryptography and Data Security*. Lecture Notes in Computer Science, Vol. 7397. Springer, 200–214.

[40] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM Computing Surveys* 41, 3, Article 15, 58 pages.

[41] Badrish Chandramouli, Jonathan Goldstein, Mike Barnett, Robert DeLine, Danyel Fisher, John C. Platt, James F. Terwilliger, and John Wernsing. 2014. Trill: A high-performance incremental query processor for diverse analytics. *Proceedings of the VLDB Endowment* 8, 4, 401–412.

[42] Badrish Chandramouli, Suman Nath, and Wenchao Zhou. 2013. Supporting distributed feed-following apps over edge devices. *Proceedings of the VLDB Endowment* 6, 13, 1570–1581.
[43] J. Chauhan, S. A. Chowdhury, and D. Makaroff. 2012. Performance evaluation of Yahoo! S4: A first look. In *Proceedings of the 2012 7th International Conference on P2P, Parallel, Grid, Cloud, and Internet Computing (3PGCIC'12)*. 58–65.
[44] Cisco Systems Inc. 2015. Cisco Video Surveillance Stream Manager Software. Available at https://www.cisco.com.
[45] Graham Cormode, Flip Korn, and Srikanta Tirthapura. 2008. Time-decaying aggregates in out-of-order streams. In *Proceedings of the 27th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'08)*. ACM, New York, NY, 89–98.
[46] Gianpaolo Cugola and Alessandro Margara. 2010. TESLA: A formally defined event specification language. In *Proceedings of the 4th ACM International Conference on Distributed Event-Based Systems (DEBS'10)*. ACM, New York, NY, 50–61.
[47] Gianpaolo Cugola and Alessandro Margara. 2012. Low latency complex event processing on parallel hardware. *Journal of Parallel and Distributed Computing* 72, 2, 205–218.
[48] Gianpaolo Cugola and Alessandro Margara. 2012. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys* 44, 3, Article 15, 62 pages.
[49] Gianpaolo Cugola and Alessandro Margara. 2013. Deployment strategies for distributed complex event processing. *Computing* 95, 2, 129–156.
[50] Gianpaolo Cugola, Alessandro Margara, Mauro Pezzè, and Matteo Pradella. 2015. Efficient analysis of event processing applications. In *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems (DEBS'15)*. ACM, New York, NY, 10–21.
[51] Michele Dallachiesa, Gabriela Jacques-Silva, Bugra Gedik, Kun-Lung Wu, and Themis Palpanas. 2015. Sliding windows over uncertain data streams. *Knowledge and Information Systems* 45, 1, 159–190.
[52] Atish Das Sarma, Sreenivas Gollapudi, and Rina Panigrahy. 2008. Estimating PageRank on graph streams. In *Proceedings of the 27th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'08)*. ACM, New York, NY, 69–78.
[53] DataTorrent. 2015. DataTorrent RTS. Retrieved January 22, 2018, from https://www.datatorrent.com/product/datatorrent-rts/.
[54] DataTorrent. 2016. DataTorrent RTS: Real-Time Streaming Analytics for Big Data. Available at https://www.datatorrent.com.
[55] Miyuru Dayarathna, Yuanlong Li, Yonggang Wen, and Rui Fan. 2017. Energy consumption analysis of data stream processing: A benchmarking approach. *Software: Practice and Experience* 47, 10, 1443–1462. DOI: http://dx.doi.org/10.1002/spe.2458 spe.2458.
[56] Miyuru Dayarathna and Toyotaro Suzumura. 2012. Hirundo: A mechanism for automated production of optimized data stream graphs. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering (ICPE'12)*. ACM, New York, NY, 335–346.
[57] Miyuru Dayarathna and Toyotaro Suzumura. 2013. Automatic optimization of stream programs via source program operator graph transformations. *Distributed and Parallel Databases* 31, 4, 543–599.
[58] Miyuru Dayarathna and Toyotaro Suzumura. 2013. A performance analysis of system S, S4, and Eper via two level benchmarking. In *Quantitative Evaluation of Systems*. Lecture Notes in Computer Science, Vol. 8054. Springer, 225–240.
[59] Gianmarco De Francisci Morales and Albert Bifet. 2015. SAMOA: Scalable Advanced Massive Online Analysis. *Journal of Machine Learning Research* 16, 1, 149–153.
[60] Xenofontas Dimitropoulos, Marc Stoecklin, Paul Hurley, and Andreas Kind. 2008. The eternal sunshine of the sketch data structure. *Computer Networks* 52, 17, 3248–3257.
[61] Arantxa Duque Barrachina and Aisling O'Driscoll. 2014. A big data methodology for categorising technical support requests using Hadoop and Mahout. *Journal of Big Data* 1, 1, 1–11. DOI: http://dx.doi.org/10.1186/2196-1115-1-1
[62] Ahmed Eldawy, Rohit Khandekar, and Kun-Lung Wu. 2012. Clustering streaming graphs. In *Proceedings of the 2012 IEEE 32nd International Conference on Distributed Computing Systems (ICDCS'12)*. IEEE, Los Alamitos,CA, 466–475.
[63] EsperTech. 2016. Performance Results. Retrieved January 22, 2018, from http://esper.espertech.com/release-5.3.0/esper-reference/html/performance.html#performance-results.
[64] EsperTech. 2015. Esper: Event Processing for Java. Retrieved January 22, 2018, from http://www.espertech.com/esper.
[65] Opher Etzion. 2009. Complex event. In *Encyclopedia of Database Systems*, L. Liu and M. Tamer Auzsul (Eds.). Springer US, 411–412.
[66] Opher Etzion, Yonit Magid, Ella Rabinovich, Inna Skarbovsky, and Nir Zolotorevsky. 2011. *Context-Based Event Processing Systems*. Springer, Berlin, Germany, 257–278. DOI: http://dx.doi.org/10.1007/978-3-642-19724-6_12

[67] Opher Etzion, Ella Rabinovich, and Inna Skarbovsky. 2011. Non functional properties of event processing. In *Proceedings of the 5th ACM International Conference on Distributed Event-Based System (DEBS'11)*. ACM, New York, NY, 365–366.

[68] Tao Feng. 2015. Benchmarking Apache Samza: 1.2 Million Messages per Second on a Single Node. Retrieved January 22, 2018, from https://engineering.linkedin.com/performance/benchmarking-apache-samza-12-million-messages-second-single-node.

[69] Ioannis Flouris, Nikos Giatrakos, Minos Garofalakis, and Antonios Deligiannakis. 2015. Issues in complex event processing systems. In *Proceedings of the 1st IEEE International Workshop on Real Time Data Stream Analytics (RT-Streams'15)*. IEEE, Los Alamitos, CA, 6.

[70] Fujitsu. 2016. FUJITSU Software Interstage Big Data Complex Event Processing Server: Benefits. Retrieved January 22, 2018, from http://www.fujitsu.com/global/products/software/middleware/application-infrastructure/interstage/solutions/big-data/bdcep/benefits/.

[71] Lajos Jenő Fülöp, Gabriella Tóth, Róbert Rácz, János Pánczél, Tamás Gergely, Árpád Beszédes, and Lóránt Farkas. 2010. Survey on complex event processing and predictive analytics. In *Proceedings of the 5th Balkan Conference in Informatics*. 26–31.

[72] Mohamed Medhat Gaber, João Gama, Shonali Krishnaswamy, João Bártolo Gomes, and Frederic Stahl. 2014. Data stream mining in ubiquitous environments: State-of-the-art and current directions. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 4, 2, 116–138.

[73] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. 2013. On evaluating stream learning algorithms. *Machine Learning* 90, 3, 317–346.

[74] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM Computing Surveys* 46, 4, Article 44, 37 pages.

[75] Feng Gao, Muhammad Intizar Ali, and Alessandra Mileo. 2014. Semantic discovery and integration of urban data streams. In *Proceedings of the 5th Workshop on Semantics for Smarter Cities: A Workshop at the 13th International Semantic Web Conference (ISWC'14)*.15–30.

[76] Minos Garofalakis, Daniel Keren, and Vasilis Samoladas. 2013. Sketch-based geometric monitoring of distributed stream queries. *Proceedings of the VLDB Endowment* 6, 10, 937–948.

[77] Buğra Gedik. 2014. Partitioning functions for stateful data parallelism in stream processing. *VLDB Journal* 23, 4, 517–539.

[78] Buğra Gedik, Rajesh R. Bordawekar, and Philip S. Yu. 2009. CellJoin: A parallel stream join operator for the cell processor. *VLDB Journal* 18, 2, 501–519.

[79] Pavlos Giakoumakis, Grigorios Chrysos, Apostolos Dollas, and Ioannis Papaefstathiou. 2015. Acceleration of data streaming classification using reconfigurable technology. In *Applied Reconfigurable Computing*. Lecture Notes in Computer Science, Vol. 9040. Springer, 357–364.

[80] Boris Glavic, Kyumars Sheykh Esmaili, Peter M. Fischer, and Nesime Tatbul. 2014. Efficient stream provenance via operator instrumentation. *ACM Transactions on Internet Technology* 14, 1, Article 7 (Aug. 2014), 26 pages.

[81] Boris Glavic, Kyumars Sheykh Esmaili, Peter Michael Fischer, and Nesime Tatbul. 2013. Ariadne: Managing fine-grained provenance on data streams. In *Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems (DEBS'13)*. ACM, New York, NY, 39–50.

[82] João Bartolo Gomes, Mohamed Medhat Gaber, Pedro A. C. Sousa, and Ernestina Menasalvas. 2013. Collaborative data stream mining in ubiquitous environments using dynamic classifier selection. *International Journal of Information Technology and Decision Making* 12, 06, 1287–1308.

[83] Michael I. Gordon, William Thies, and Saman Amarasinghe. 2006. Exploiting coarse-grained task, data, and pipeline parallelism in stream programs. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XII)*. ACM, New York, NY, USA, 151–162.

[84] Torsten Grabs and Ming Lu. 2012. Measuring performance of complex event processing systems. In *Topics in Performance Evaluation, Measurement and Characterization*. Lecture Notes in Computer Science, Vol. 7144. Springer, 83–96.

[85] Jamie Grier. 2016. Extending the Yahoo! Streaming Benchmark. Retrieved January 22, 2018, from https://data-artisans.com/blog/extending-the-yahoo-streaming-benchmark.

[86] Yu Gu, Zhe Zhang, Fan Ye, Hao Yang, Minkyong Kim, Hui Lei, and Zhen Liu. 2009. An empirical study of high availability in stream processing systems. In *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware (Middleware'09)*. Article 23, 9 pages.

[87] Sudipto Guha, Andrew McGregor, and David Tench. 2015. Vertex and hyperedge connectivity in dynamic graph streams. In *Proceedings of the 34th ACM Symposium on Principles of Database Systems (PODS'15)*. ACM, New York, NY, 241–247.

[88] V. Gulisano, R. Jimenez-Peris, M. Patino-Martinez, and P. Valduriez. 2010. StreamCloud: A large scale data streaming system. In *Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems (ICDCS'10)*. 126–137.

[89] Jagabondhu Hazra, Kaushik Das, Deva P. Seetharam, and Amith Singhee. 2011. Stream computing based synchrophasor application for power grids. In *Proceedings of the 1st International Workshop on High Performance Computing, Networking, and Analytics for the Power Grid (HiPCNA-PG'11)*. ACM, New York, NY, 43–50.

[90] Bingsheng He, Mao Yang, Zhenyu Guo, Rishan Chen, Bing Su, Wei Lin, and Lidong Zhou. 2010. Comet: Batched stream processing for data intensive distributed computing. In *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC'10)*. ACM, New York, NY, 63–74.

[91] Thomas Heinze, Leonardo Aniello, Leonardo Querzoni, and Zbigniew Jerzak. 2014. Cloud-based data stream processing. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems (DEBS'14)*. ACM, New York, NY, 238–245.

[92] Thomas Heinze, Valerio Pappalardo, Zbigniew Jerzak, and Christof Fetzer. 2014. Auto-scaling techniques for elastic data stream processing. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems (DEBS'14)*. ACM, New York, NY, 318–321.

[93] Thomas Heinze, Lars Roediger, Andreas Meister, Yuanzhen Ji, Zbigniew Jerzak, and Christof Fetzer. 2015. Online parameter optimization for elastic data stream processing. In *Proceedings of the 6th ACM Symposium on Cloud Computing (SoCC'15)*. ACM, New York, NY, 276–287.

[94] W. A. Higashino, M. A. M. Capretz, and L. F. Bittencourt. 2015. CEPSim: A simulator for cloud-based complex event processing. In *Proceedings of the 2015 IEEE International Congress on Big Data (BigData Congress'15)*. 182–190.

[95] Matthew Hill, Murray Campbell, Yuan-Chi Chang, and Vijay Iyengar. 2008. Event detection in sensor networks for modern oil fields. In *Proceedings of the 2nd International Conference on Distributed Event-Based Systems (DEBS'08)*. ACM, New York, NY, 95–102.

[96] Annika Hinze, Kai Sachs, and Alejandro Buchmann. 2009. Event-based applications and enabling technologies. In *Proceedings of the 3rd ACM International Conference on Distributed Event-Based Systems (DEBS'09)*. ACM, New York, NY, Article 1, 15 pages. DOI: http://dx.doi.org/10.1145/1619258.1619260

[97] Martin Hirzel. 2012. Partition and compose: Parallel complex event processing. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems (DEBS'12)*. ACM, New York, NY, 191–200.

[98] M. Hirzel, H. Andrade, B. Gedik, G. Jacques-Silva, R. Khandekar, V. Kumar, M. Mendell, H. Nasgaard, S. Schneider, R. Soule, and K.-L. Wu. 2013. IBM streams processing language: Analyzing big data in motion. *IBM Journal of Research and Development* 57, 3-4, 7:1–7:11.

[99] Martin Hirzel, Rodric Rabbah, Philippe Suter, Olivier Tardieu, and Mandana Vaziri. 2015. Spreadsheets for stream partitions and windows. In *Proceedings of the 2nd Workshop on Software Engineering Methods in Spreadsheets Co-Located With the 37th International Conference on Software Engineering (ICSE'15)*. 39–40.

[100] Martin Hirzel, Robert Soulé, Scott Schneider, Buğra Gedik, and Robert Grimm. 2014. A catalog of stream processing optimizations. *ACM Computing Surveys* 46, 4, Article 46, 34 pages.

[101] T. Ryan Hoens, Robi Polikar, and Nitesh V. Chawla. 2012. Learning from streaming data with concept drift and imbalance: An overview. *Progress in Artificial Intelligence* 1, 1, 89–101.

[102] Geoff Hulten and Pedro Domingos. 2003. VFM—A Toolkit for Mining High-Speed Time-Changing Data Streams. Retrieved January 22, 2018, from http://www.cs.washington.edu/dm/vfml/.

[103] Jeong-Hyon Hwang, Magdalena Balazinska, Alexander Rasin, Ugur Cetintemel, Michael Stonebraker, and Stan Zdonik. 2005. High-availability algorithms for distributed stream processing. In *Proceedings of the 21st International Conference on Data Engineering (ICDE'05)*. IEEE, Los Alamitos, CA, 779–790.

[104] A. Ishii and T. Suzumura. 2011. Elastic stream computing with clouds. In *Proceedings of the 2011 IEEE International Conference on Cloud Computing (CLOUD'11)*. 195–202.

[105] Sachini Jayasekara, Srinath Perera, Miyuru Dayarathna, and Sriskandarajah Suhothayan. 2015. Continuous analytics on geospatial data streams with WSO2 complex event processor. In *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems (DEBS'15)*. ACM, New York, NY, 277–284.

[106] Zbigniew Jerzak and Holger Ziekow. 2015. The DEBS 2015 grand challenge. In *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems (DEBS'15)*. ACM, New York, NY, 266–268.

[107] Yuanzhen Ji. 2013. Database support for processing complex aggregate queries over data streams. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops (EDBT'13)*. ACM, New York, NY, 31–37.

[108] Yuanzhen Ji, Hongjin Zhou, Zbigniew Jerzak, Anisoara Nica, Gregor Hackenbroich, and Christof Fetzer. 2015. Quality-driven processing of sliding window aggregates over out-of-order data streams. In *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems (DEBS'15)*. ACM, New York, NY, 68–79.

[109] Tomas Karnagel, Dirk Habich, Benjamin Schlegel, and Wolfgang Lehner. 2013. The HELLS-join: A heterogeneous stream join for extremely large windows. In *Proceedings of the 9th International Workshop on Data Management on New Hardware (DaMoN'13)*. ACM, New York, NY, Article 2, 7 pages.

[110] Tomas Karnagel, Benjamin Schlegel, Dirk Habich, and Wolfgang Lehner. 2013. Stream join processing on heterogeneous processors. In *Proceedings of the 2013 BTW Workshops*. 17–26.

[111] Rohit Khandekar, Kirsten Hildrum, Sujay Parekh, Deepak Rajan, Joel Wolf, Kun-Lung Wu, Henrique Andrade, and Buğra Gedik. 2009. COLA: Optimizing stream processing applications via graph partitioning. In *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware (Middleware'09)*. Article 16, 20 pages.

[112] Arush Kharbanda. 2015. Fault Tolerant Stream Processing With Spark Streaming. Retrieved January 22, 2018, from https://www.sigmoid.com/fault-tolerant-stream-processing/.

[113] W. Kleiminger, E. Kalyvianaki, and P. Pietzuch. 2011. Balancing load in stream processing with the cloud. In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering Workshops (ICDEW'11)*. 16–21.

[114] Lazaros Koromilas, Giorgos Vasiliadis, Ioannis Manousakis, and Sotiris Ioannidis. 2014. Efficient software packet processing on heterogeneous and asymmetric hardware architectures. In *Proceedings of the 10th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS'14)*. ACM, New York, NY, 207–218.

[115] Sailesh Krishnamurthy, Michael J. Franklin, Jeffrey Davis, Daniel Farina, Pasha Golovko, Alan Li, and Neil Thombre. 2010. Continuous analytics over discontinuous streams. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data (SIGMOD'10)*. ACM, New York, NY, 1081–1092.

[116] S. P. T. Krishnan and J. L. Ugia Gonzalez. 2015. Google cloud dataflow. In *Building Your Next Big Thing With Google Cloud Platform*. Apress, New York, NY, 255–275.

[117] Mino Ku, Eunmi Choi, and Dugki Min. 2014. An analysis of performance factors on Esper-based stream big data processing in a virtualized environment. *International Journal of Communication Systems* 27, 6, 898–917.

[118] Sanjeev Kulkarni, Nikunj Bhagat, Maosong Fu, Vikas Kedigehalli, Christopher Kellogg, Sailesh Mittal, Jignesh M. Patel, Karthik Ramasamy, and Siddarth Taneja. 2015. Twitter Heron: Stream processing at scale. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD'15)*. ACM, New York, NY, 239–250.

[119] Kx Systems Inc. 2016. Real Time Analytics, Kx Capabilities, Kx Systems. Retrieved January 22, 2018, from https://kx.com/real-time-in-memory-analytics.php.

[120] Danh Le-Phuoc, Minh Dao-Tran, Minh-Duc Pham, Peter Boncz, Thomas Eiter, and Michael Fink. 2012. Linked stream data processing engines: Facts and figures. In *Proceedings of the 11th International Conference on the Semantic Web, Part II (ISWC'12)*. 300–312.

[121] Boduo Li, Yanlei Diao, and Prashant Shenoy. 2015. Supporting scalable analytics with latency constraints. *Proceedings of the VLDB Endowment* 8, 1, 1166–1177.

[122] Erietta Liarou, Stratos Idreos, Stefan Manegold, and Martin Kersten. 2013. Enhanced stream processing in a DBMS kernel. In *Proceedings of the 16th International Conference on Extending Database Technology (EDBT'13)*. ACM, New York, NY, 501–512.

[123] M. Liu, M. Li, D. Golovnya, E. A. Rundensteiner, and K. Claypool. 2009. Sequence pattern query processing over out-of-order event streams. In *Proceedings of the 2009 IEEE 25th International Conference on Data Engineering (ICDE'09)*. 784–795.

[124] Björn Lohrmann, Daniel Warneke, and Odej Kao. 2012. Massively-parallel stream processing under QoS constraints with Nephele. In *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing (HPDC'12)*. ACM, New York, NY, 271–282.

[125] David Luckham. 2016. Proliferation of Open Source Technology for Event Processing. Retrieved January 22, 2018, from http://www.complexevents.com/2016/06/15/proliferation-of-open-source-technology-for-event-processing/.

[126] K. F. Lysakov and M. Y. Shadrin. 2013. FPGA-based hardware accelerator for high-performance data-stream processing. *Pattern Recognition and Image Analysis* 23, 1, 26–34.

[127] Mahmoud S. Mahmoud, Andrew Ensor, Alain Biem, Bruce Elmegreen, and Sergei Gulyaev. 2013. Data provenance and management in radio astronomy: A stream computing approach. In *Data Provenance and Data Management in eScience*. Studies in Computational Intelligence, Vol. 426. Springer, 129–156.

[128] Na Mao and Jie Tan. 2015. Complex event processing on uncertain data streams in product manufacturing process. In *Proceedings of the 2015 International Conference on Advanced Mechatronic Systems (ICAMechS'15)*. 583–588.

[129] Alessandro Margara and Gianpaolo Cugola. 2014. High-performance publish-subscribe matching using parallel hardware. *IEEE Transactions on Parallel and Distributed Systems* 25, 1, 126–135.

[130] André Martin, Andrey Brito, and Christof Fetzer. 2014. Scalable and elastic realtime click stream analysis using StreamMine3G. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems (DEBS'14)*. ACM, New York, NY, 198–205.

[131] André Martin, Andrey Brito, and Christof Fetzer. 2015. Real time data analysis of taxi rides using StreamMine3G. In *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems (DEBS'15)*. ACM, New York, NY, 269–276.

[132] Henry May, David Engebretsen, and Walt Madden. 2016. IBM InfoSphere Streams v4.0 Performance Best Practices. Retrieved January 22, 2018, from https://developer.ibm.com/streamsdev/wp-content/uploads/sites/15/2015/04/Streams_4.0.0.0_PerformanceBestPractices.pdf.

[133] Andrew McGregor. 2014. Graph stream algorithms: A survey. *ACM SIGMOD Record* 43, 1, 9–20.
[134] Gèrard Medioni, Isaac Cohen, François Brèmond, Somboon Hongeng, and Ramakant Nevatia. 2001. Event detection and analysis from video streams. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 23, 8, 873–889.
[135] John Meehan, Nesime Tatbul, Stanley B. Zdonik, Cansu Aslantas, Ugur Çetintemel, Jiang Du, Tim Kraska, et al. 2015. S-Store: Streaming meets transaction processing. arXiv:1503.01143.
[136] Marcelo R. Mendes, Pedro Bizarro, and Paulo Marques. 2009. A performance study of event processing systems. In *Performance Evaluation and Benchmarking*. Springer-Verlag, Berlin, Germany, 221–236.
[137] Marcelo R. N. Mendes, Pedro Bizarro, and Paulo Marques. 2013. FINCoS: Benchmark tools for event processing systems. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering (ICPE'13)*. ACM, New York, NY, 431–432.
[138] Xiangrui Meng, Joseph K. Bradley, Burak Yavuz, Evan R. Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, et al. 2015. MLlib: Machine learning in Apache Spark. arXiv:1505.06807.
[139] Zachary Miller, Brian Dickinson, William Deitrick, Wei Hu, and Alex Hai Wang. 2014. Twitter spammer detection using data stream clustering. *Information Sciences* 260, 64–73.
[140] Archan Misra, Marion Blount, Anastasios Kementsietsidis, Daby Sow, and Min Wang. 2008. Advances and challenges for scalable provenance in stream processing systems. In *Provenance and Annotation of Data and Processes*. Lecture Notes in Computer Science, Vol. 5272. Springer, 253–265.
[141] Rene Mueller, Jens Teubner, and Gustavo Alonso. 2009. Streams on wires: A query compiler for FPGAs. *Proceedings of the VLDB Endowment* 2, 1, 229–240.
[142] Gero Mühl, Ludger Fiege, and Alejandro P. Buchmann. 2002. Filter similarities in content-based publish/subscribe systems. In *Proceedings of the International Conference on Architecture of Computing Systems: Trends in Network and Pervasive Computing (ARCS'02)*. 224–240. http://dl.acm.org/citation.cfm?id=648198.751352.
[143] Derek G. Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martín Abadi. 2013. Naiad: A timely dataflow system. In *Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP'13)*. ACM, New York, NY, 439–455.
[144] Christopher Mutschler and Michael Philippsen. 2014. Adaptive speculative processing of out-of-order event streams. *ACM Transactions on Internet Technology* 14, 1, Article 4, 24 pages.
[145] Zubair Nabi, Eric Bouillet, Andrew Bainbridge, and Chris Thomas. 2014. *Of Streams and Storms. White Paper. IBM.*
[146] Roshan Naik and Sapin Amin. 2016. Microbenchmarking Apache Storm 1.0 Performance. Retrieved January 22, 2018, from http://hortonworks.com/blog/microbenchmarking-storm-1-0-performance/.
[147] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari. 2010. S4: Distributed stream computing platform. In *Proceedings of the 2010 IEEE International Conference on Data Mining Workshops (ICDMW'10)*. 170–177.
[148] Odysseas Papapetrou, Minos Garofalakis, and Antonios Deligiannakis. 2012. Sketch-based querying of distributed sliding-window data streams. *Proceedings of the VLDB Endowment* 5, 10, 992–1003.
[149] A. Pavan, K. Tangwongsan, S. Tirthapura, and K.-L. Wu. 2013. Counting and sampling triangles from a graph stream. *Proceedings of the VLDB Endowment* 6, 14, 1870–1881.
[150] Srinath Perera. 2013. CEP Performance: Processing 100k to Millions of Events per Second Using WSO2 Complex Event Processing (CEP) Server. Retrieved January 22, 2018, from http://srinathsview.blogspot.com/2013/08/cep-performance-processing-100k-to.html.
[151] Srinath Perera, Suhothayan Sriskandarajah, Mohanadarshan Vivekanandalingam, Paul Fremantle, and Sanjiva Weerawarana. 2014. Solving the grand challenge using an opensource CEP engine. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems (DEBS'14)*. ACM, New York, NY, 288–293. DOI: http://dx.doi.org/10.1145/2611286.2611331
[152] Niko Pollner, Christian Steudtner, and Klaus Meyer-Wegener. 2015. Placement-safe operator-graph changes in distributed heterogeneous data stream systems. In *Datenbanksysteme für Business, Technologie, und Web (BTW 2015)*. 61–70.
[153] Zhengping Qian, Yong He, Chunzhi Su, Zhuojie Wu, Hongyu Zhu, Taizhi Zhang, Lidong Zhou, Yuan Yu, and Zheng Zhang. 2013. TimeStream: Reliable stream computation in the cloud. In *Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys'13)*. ACM, New York, NY, 1–14.
[154] Ella Rabinovich, Opher Etzion, and Avigdor Gal. 2011. Pattern rewriting framework for event processing optimization. In *Proceedings of the 5th ACM International Conference on Distributed Event-Based System (DEBS'11)*. ACM, New York, NY, 101–112.
[155] Chris Raphael. 2014. IDC Reveals Worldwide Internet of Things Predictions for 2015. Retrieved January 22, 2018, from https://blog.ipswitch.com/2015-predictions-the-impact-of-the-internet-of-things-on-the-network.
[156] Chris Raphael. 2015. IoT Architectures for Edge Analytics. Retrieved January 22, 2018, from http://www.rtinsights.com/iot-architectures-for-edge-analytics/.

[157] Sajith Ravindra and Miyuru Dayarathna. 2015. Distributed Scaling of WSO2 Complex Event Processor. Retrieved January 22, 2018, from http://wso2.com/library/articles/2015/12/article-distributed-scaling-of-wso2-complex-event-processor/.
[158] Vlad Rozov. 2015. Apache Apex Performance Benchmarks. Retrieved January 22, 2018, from https://www.datatorrent.com/blog/blog-apex-performance-benchmark/.
[159] Vlad Rozov. 2015. Apache Apex Performance Benchmarks. Retrieved January 22, 2018, from https://www.datatorrent.com/blog/blog-apex-performance-benchmark/.
[160] SourceForge. 2015. Complex Event Pattern Detector. Retrieved January 22, 2018, from http://sourceforge.net/projects/rulecore/.
[161] Florin Rusu and Alin Dobra. 2007. Statistical analysis of sketch estimators. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data (SIGMOD'07)*. ACM, New York, NY, 187–198.
[162] S. Chintapalli, D. Dagit, B. Evans, R. Farivar, T. Graves, M. Holderbaugh, Z. Liu, et al. 2015. Benchmarking Streaming Computation Engines at Yahoo! Retrieved January 22, 2018, from https://yahooeng.tumblr.com/post/135321837876/benchmarking-streaming-computation-engines-at.
[163] Ivo Santos, Marcel Tilly, Badrish Chandramouli, and Jonathan Goldstein. 2013. DiAl: Distributed streaming analytics anywhere, anytime. *Proceedings of the VLDB Endowment* 6, 12, 1386–1389.
[164] Ahmet Erdem Saríyüce, Buğra Gedik, Gabriela Jacques-Silva, Kun-Lung Wu, and Ümit V. Çatalyürek. 2013. Streaming algorithms for K-Core decomposition. *Proceedings of the VLDB Endowment* 6, 6, 433–444.
[165] Scott Schneider, Martin Hirzel, Bugra Gedik, and Kun-Lung Wu. 2012. Auto-parallelizing stateful distributed streaming applications. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT'12)*. ACM, New York, NY, 53–64.
[166] W. R. Schulte. 2015. CEP Technology: EPPs, DSCPs and Other Product Categories. Retrieved January 22, 2018, from http://www.complexevents.com/2015/07/10/cep-technology-epps-dscps-and-other-product-categories.
[167] Nicholas Poul Schultz-Møller, Matteo Migliavacca, and Peter Pietzuch. 2009. Distributed complex event processing with query rewriting. In *Proceedings of the 3rd ACM International Conference on Distributed Event-Based Systems (DEBS'09)*. ACM, New York, NY, Article 4, 12 pages.
[168] Izchak Sharfman, Assaf Schuster, and Daniel Keren. 2006. A geometric approach to monitoring threshold functions over distributed data streams. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data (SIGMOD'06)*. ACM, New York, NY, 301–312.
[169] Shohei Hido, Seiya Tokui, and Satoshi Oda. 2013. Jubatus: An open source platform for distributed online machine learning. In *Proceedings of the Big Learning Workshop at Advances in Neural Information Processing Systems 26 (NIPS'13)*. 6.
[170] Jonathan A. Silva, Elaine R. Faria, Rodrigo C. Barros, Eduardo R. Hruschka, André C. P. L. F. de Carvalho, and João Gama. 2013. Data stream clustering: A survey. *ACM Computing Surveys* 46, 1, Article 13, 31 pages.
[171] Yogesh Simmhan, Baohua Cao, Michail Giakkoupis, and Viktor K. Prasanna. 2011. Adaptive rate stream processing for smart grid applications on clouds. In *Proceedings of the 2nd International Workshop on Scientific Cloud Computing (ScienceCloud'11)*. ACM, New York, NY, 33–38.
[172] Anastasios Skarlatidis, Alexander Artikis, Jason Filippou, and Georgios Paliouras. 2015. A probabilistic logic programming event calculus. *Theory and Practice of Logic Programming* 15, 2, 213–245.
[173] Chunyao Song, Tingjian Ge, Cindy Chen, and Jie Wang. 2014. Event pattern matching over graph streams. *Proceedings of the VLDB Endowment* 8, 4, 413–424.
[174] Robert Soulé, Martin Hirzel, Buğra Gedik, and Robert Grimm. 2012. From a calculus to an execution environment for stream processing. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems (DEBS'12)*. ACM, New York, NY, 20–31.
[175] Robert Soule, Martin Hirzel, Bugra Gedik, and Robert Grimm. 2015. River: An intermediate language for stream processing. *Software: Practice and Experience* 46, 7, 891–929.
[176] SQLstream. 2014. SQLstream Blaze and Apache Storm: A Benchmark Comparison. Retrieved January 22, 2018, from http://www.sqlstream.com/.
[177] SQLstream. 2015. Home Page. Retrieved January 22, 2018, from http://www.sqlstream.com/.
[178] SQLstream. 2016. SQLstream Blaze Outperforms Apache Storm in Stream Processing Benchmark. Retrieved January 22, 2018, from http://sqlstream.com/2014/11/sqlstream-blaze-over-100x-faster-than-apache-storm-in-industry-benchmark-for-stream-processing-performance/.
[179] Utkarsh Srivastava and Jennifer Widom. 2004. Flexible time management in data stream systems. In *Proceedings of the 23rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'04)*. ACM, New York, NY, 263–274.
[180] Isabelle Stanton and Gabriel Kliot. 2012. Streaming graph partitioning for large distributed graphs. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'12)*. ACM, New York, NY, 1222–1230.

[181] Sriskandarajah Suhothayan, Kasun Gajasinghe, Isuru Loku Narangoda, Subash Chaturanga, Srinath Perera, and Vishaka Nanayakkara. 2011. Siddhi: A second look at complex event processing architectures. In *Proceedings of the 2011 ACM Workshop on Gateway Computing Environments (GCE'11)*. ACM, New York, NY, 43–50.

[182] Dawei Sun, Guangyan Zhang, Songlin Yang, Weimin Zheng, Samee U. Khan, and Keqin Li. 2015. Re-Stream: Real-time and energy-efficient resource scheduling in big data stream computing environments. *Information Sciences* 319, 2015, 92–112.

[183] Dawei Sun, Guangyan Zhang, Weimin Zheng, and Keqin Li. 2015. Key technologies for big data stream computing. In *Big Data: Algorithms, Analytics, and Applications*. CRC Press, Boca Raton, FL, 193–214.

[184] Bart Theeten, Ivan Bedini, Peter Cogan, Alessandra Sala, and Tommaso Cucinotta. 2014. Towards the optimization of a parallel streaming engine for Telco applications. *Bell Labs Technical Journal* 18, 4, 181–197.

[185] Richard Tibbetts. 2009. *Performance and Scalability Characterization*. StreamBase.

[186] TIBCO. 2014. TIBCO StreamBase Versus Native Threading. Retrieved January 22, 2018, from https://d2wh20haedxe3f.cloudfront.net/sites/default/files/wiki_files/wp-tibco-streambase-versus-native-threading.

[187] TIBCO. 2016. StreamBase Studio. Available at http://www.streambase.com/products/streambasecep/streambase-studio/.

[188] Srikanta Tirthapura, Bojian Xu, and Costas Busch. 2006. Sketching asynchronous streams over a sliding window. In *Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing (PODC'06)*. ACM, New York, NY, 82–91.

[189] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M. Patel, Sanjeev Kulkarni, Jason Jackson, et al. 2014. Storm@Twitter. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD'14)*. ACM, New York, NY, 147–156.

[190] Thanh T. Tran, Liping Peng, Yanlei Diao, Andrew McGregor, and Anna Liu. 2012. CLARO: Modeling and processing uncertain data streams. *VLDB Journal* 21, 5, 651–676.

[191] Radu Tudoran, Olivier Nano, Ivo Santos, Alexandru Costan, Hakan Soncu, Luc Bougé, and Gabriel Antoniu. 2014. JetStream: Enabling high performance event streaming across cloud data-centers. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems (DEBS'14)*. ACM, New York, NY, 23–34.

[192] Kostas Tzoumas, Stephan Ewen, and Robert Metzger. 2015. High-Throughput, Low-Latency, and Exactly-Once Stream Processing With Apache Flink. Retrieved January 22, 2018, from https://data-artisans.com/high-throughput-low-latency-and-exactly-once-stream-processing-with-apache-flink/.

[193] Uri Verner, Assaf Schuster, and Mark Silberstein. 2011. Processing data streams with hard real-time constraints on heterogeneous systems. In *Proceedings of the International Conference on Supercomputing (ICS'11)*. ACM, New York, NY, 120–129.

[194] Paul Vincent. 2016. CEP Tooling Market Survey 2016. Retrieved January 22, 2018, from http://www.complexevents.com/2016/05/12/cep-tooling-market-survey-2016/.

[195] Y. Wang and K. Cao. 2012. Context-aware complex event processing for event cloud in Internet of Things. In *Proceedings of the 2012 International Conference on Wireless Communications Signal Processing (WCSP'12)*. 1–6. DOI: http://dx.doi.org/10.1109/WCSP.2012.6542861

[196] Y. H. Wang, K. Cao, and X. M. Zhang. 2013. Complex event processing over distributed probabilistic event streams. *Computers and Mathematics With Applications* 66, 10, 1808–1821.

[197] Zheng Wang and Michael F. P. O'Boyle. 2010. Partitioning streaming parallelism for multi-cores: A machine learning based approach. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques (PACT'10)*. ACM, New York, NY, 307–318.

[198] Jielong Xu, Zhenhua Chen, Jian Tang, and Sen Su. 2014. T-Storm: Traffic-aware online scheduling in storm. In *Proceedings of the 2014 IEEE 34th International Conference on Distributed Computing Systems (ICDCS'14)*. 535–544.

[199] A. Yamaguchi, Y. Nakamoto, K. Sato, Y. Ishikawa, Y. Watanabe, S. Honda, and H. Takada. 2015. AEDSMS: Automotive embedded data stream management system. In *Proceedings of the 2015 IEEE 31st International Conference on Data Engineering (ICDE'15)*. 1292–1303.

[200] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. 2013. Discretized streams: Fault-tolerant streaming computation at scale. In *Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP'13)*. ACM, New York, NY, 423–438.

[201] Erik Zeitler and Tore Risch. 2011. Massive scale-out of expensive continuous queries. *Proceedings of the VLDB Endowment* 4, 11, 1181–1188.

[202] Chunwang Zhang and Ee Chien Chang. 2014. Processing of mixed-sensitivity video surveillance streams on hybrid clouds. In *Proceedings of the 2014 IEEE 7th International Conference on Cloud Computing (CLOUD'14)*. 9–16.

[203] Haopeng Zhang, Yanlei Diao, and Neil Immerman. 2010. Recognizing patterns in streams with imprecise timestamps. *Proceedings of the VLDB Endowment* 3, 1–2, 244–255.

[204] Haopeng Zhang, Yanlei Diao, and Neil Immerman. 2014. On complexity and optimization of expensive queries in complex event processing. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD'14)*. ACM, New York, NY, 217–228.

[205] Zhe Zhang, Yu Gu, Fan Ye, Hao Yang, Minkyong Kim, Hui Lei, and Zhen Liu. 2010. A hybrid approach to high availability in stream processing systems. In *Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems (ICDCS'10)*. 138–148.

[206] Qunzhi Zhou, Yogesh Simmhan, and Viktor Prasanna. 2012. Incorporating semantic knowledge into dynamic data processing for smart power grids. In *The Semantic Web*. Lecture Notes in Computer Science, Vol. 7650. Springer, 257–273.

Received May 2016; revised November 2016; accepted December 2017