

Recent Advancements in Event Processing

MIYURU DAYARATHNA, WSO2, Inc., USA
SRINATH PERERA, WSO2, Inc., USA

Event Processing (EP) is a data processing technology which 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 view points. We divide the entire field of event processing into three sub areas as EP system architectures, EP use cases, and EP Open Research topics. Then we deep dive into the details of each subsection. We investigate on system architecture characteristics of novel event processing platforms such as Apache Storm, Apache Spark, Apache Flink, etc. We found significant advancements made on novel application areas such as Internet of Things (IoT), Streaming Machine Learning, processing of complex data types such as text, video data streams, and graphs. Furthermore, there have been significant body of contributions made on event ordering, system scalability, development of event processing languages, and exploration of use of heterogeneous devices for EP which we investigate in the latter half of this paper. Through our study we found key areas which require significant attention from EP community such as Streaming ML, EP system benchmarking, graph stream processing, etc.

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. 2017. Recent Advancements in Event Processing. *ACM Comput. Surv.* 1, 1, Article 1 (January 2017), 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. A number of real world applications can be found with the requirement of processing information which are flowing from outside to the system [48]. Vast number of recent applications of EP could be found in the areas of health informatics [31], astronomy [127], telecommunications [184], electric grids [89] and energy [95], geography, transportation [105], etc. 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, different types of windows, etc. A subset of EP applications (see Figure 1) may involve detecting complex events such as anomaly detection, predictive analytics, etc. which are termed as complex event processing (CEP).

Authors' addresses: Miyuru Dayarathna, WSO2, Inc. 787, Castro Street, Mountain View, CA, 94041, USA, miyurud@wso2.com; Srinath Perera, WSO2, Inc. 787, Castro Street, Mountain View, CA, 94041, USA, srinath@wso2.com.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2017 Association for Computing Machinery.

0360-0300/2017/1-ART1 \$15.00

<https://doi.org/10.1145/3170432>

EP has gained a significant market share in data processing technologies. Two recent articles forecast that Complex Event Processing (CEP) market will grow from \$1005 million to \$4762 million in 2019 [2] while 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 (CAGR) of 31.3% and 36.5% respectively.

The roots of EP can be traced back to the era of Active Databases [7]. Since then, number of EP software systems have appeared. EP can be divided into two main areas called Stream processing and Complex Event Processing (CEP). The first area, Stream (i.e., event) processing supports many kinds of continuous analytics such as filter, aggregation, enrichment, classification, joining, etc. CEP on the other hand 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 recent works CEP has been implemented as an operator in a general-purpose stream processing system [97]. On the other hand CEP engines have implemented support for conducting general purpose event processing. We believe that CEP can be considered as a special form of event processing, hence the theories and applications devised for event processing are equally applicable to CEP as well (See Figure 1 (a)). A recent article by Schulte, categorized EP system as Event Stream Processing (ESP) platforms, Event Processing Platforms (EPPs), Distributed Stream Computing Platforms (DSCPs), and Complex-event processing (CEP) platforms or engines [166]. In this paper we adapt the notion in categorizing EP software systems described in [166] (See Figure 1 (b)).

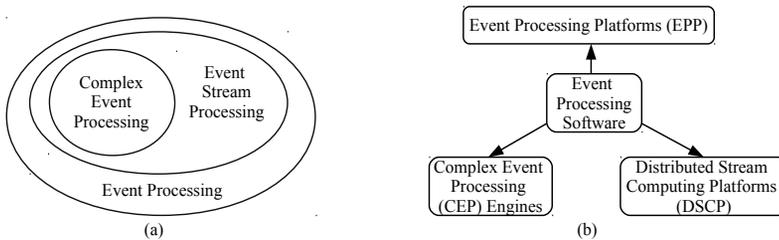


Fig. 1. Event Processing Landscape. (a) Difference between event processing terminologies. (b) Types of EP software. In this paper we classify the entire domain of even processing software into three areas CEP Engines, EPPs, and DSCPs.

While there have been few surveys conducted on EP systems with the most recent one published almost half a decade ago, a significant body of research have been conducted in the realm of EP systems. New distributed stream computing platforms have been introduced recently. New use cases of EP have arrived while technology advancements have been made relating to the quality of service characteristics of event processing systems. However, no recent survey being 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 to 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, autotuning of EP query networks, EP provenance, EP benchmarking efforts, EP query languages, etc.

2 RELATED SURVEYS

There have been few detailed surveys conducted in the context of EP systems. The surveys published till now can be categorized into three main areas: EP in General, Streaming Algorithms, and Stream Processing Optimizations.

Table 1. Comparison of related surveys.

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

Similar to our work, majority of the 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.* examined terminology, research achievements, existing solutions, and open issues of CEP [71]. They have provided an introduction to innovative papers on CEP, basic concepts of CEP, and a detailed taxonomy of the CEP tools. Furthermore, they have conducted a detailed assessment of the technologies of Predictive Analytics. The survey made by Cugola *et al.* circa 2012 unified data stream processing and complex event processing under a single term called Information Flow Processing (IFP) [48]. They claimed that neither data stream processing nor complex event processing models could completely function as an IFP engine which consists of characteristics of the two areas. Similar to us, they drew a framework which is general enough for comparisons. They used this framework and made comprehensive investigation of the cutting edge. However, the field of event processing has undergone significant developments since Cugola *et al.*'s work has been published. For example, the field of the second generation distributed stream processing systems (DSCP) has been developed in last five - seven years time. Furthermore, novel application areas of event processing have emerged such as IoT, streaming machine learning, etc. Hence, we believe it is important to investigate on these recent trends in event processing. In a more recent survey, Flouris *et al.* made an overview of the main research issues faced by existing CEP technologies [69]. They created a taxonomy of CEP performance optimization approaches highlighting main areas as Predicate-Related Optimizations, Buffering techniques, and Query Rewriting/Reordering. Furthermore, they discussed about 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 as EP use cases, EP system architectures, and EP open research topics (See Figure 1 (b)). In another recent survey, Sun *et al.* investigated on Big Data Stream Processing (BDSP) from the point of view of the systems which are used for BDSP [183]. In their work first they provided a general overview to 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.* was on stream processing on ubiquitous environments [72]. Different from both these works, we conduct the survey with much more broader scope.

There are few notable surveys being conducted on the algorithmic aspects of stream processing recently. Stream clustering has been an area of great importance. The work done by Silva *et al.* is one of the earliest examples of this category of surveys. Another example is the work done by Aggarwal [11]. The work by Amini *et al.* [17] is the third example for such surveys. Graph stream algorithms is another category of stream processing algorithms which attracted significant attention in recent times. The survey published by McGregor highlights multiple work conducted in this area [133]. However, in this paper we survey the area from the perspective of graph stream processing applications.

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

A chronologically ordered list of the aforementioned surveys are listed in Table 1. In this paper we focus on EP in general and discuss the latest trends in EP. We believe the comprehensive coverage made on latest state-of-the-art research on EP will be of great interest for EP community.

3 OVERVIEW OF THE EVENT PROCESSING

An event is simply a real world incident which has been captured by a system. A complex event is an event derived from a group of events using either aggregation or 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 which captures and processes the occurrence of real world incidents. CEP can be defined as computing which conducts operations on complex events. [166].

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, video streams collected from surveillance cameras, etc. The EP engine is the brain of the processing which does multiple types of processing on event streams based on predefined rules. The processing includes simple counting, averaging, median calculation type of simple event processing operations as well as more complex processing such as pattern matching, event prediction (forecasting), etc. Event consumers are parties who are interested of mining valuable information from the event streams. E.g., software agents, users of web/mobile applications, etc.

In this paper we divide the advancements made on EP into three sub-categories as *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. Internet of Things (IoT), data stream processing with complex data types such as text, video, graphs, etc. are some examples for this category. In this paper we describe recent EP use cases from the point-of-view of general technological paradigms. For example, while 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 [96]. We categorize the improvements made on the operational aspects of EP systems such as high availability and fault tolerance, security, correctness, performance and scalability, etc. under EP open research topics. It should be noted that although we have made three categories as mentioned above, in a particular advancement made on EP there can be several of these categories interlinked together.

4 EVENT PROCESSING SYSTEM ARCHITECTURES

The implementation of EP concepts are made in the context of Event Stream Processing systems (ESPs). The software tools used for EP can be categorized under three sub types as Event Processing Platforms [166], Distributed Stream Computing Platforms, and CEP Libraries (i.e., CEP Engines). This section elaborates more on these sub topics. Event Processing Platforms (EPP) are Event Stream Processing software that provide high-level programming models such as expressive event processing languages (EPLs) and built-in functions for event filtering, correlation, and abstraction. Distributed Stream Computing Platforms (DSCPs) (Also can be termed as *Distributed Stream Processing Engines (DSPEs)* [19]) are Event Stream Processing Platforms that provide explicit support for distribution of computation across multiple nodes in a compute cluster. CEP library (we also use the term CEP engine interchangeably) in general is a software component which specially focuses on detection of complex events.

4.1 Event Processing Platforms

IBM InfoSphere Streams [24], DataTorrent Real-time Streaming (RTS) [53] (Apache Apex), SQL-Stream s-Server [177], WSO2 CEP [105], StreamMine3G [131], etc. are some notable examples for EPPs. Note that EPPs can be developed following publish/subscribe pattern. In such systems generally events 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 event processing system.

A summary of feature comparison between different event processing platforms is shown in Table 2. Note that if a feature is present we mark it using a black bullet (●) while the missing features are marked using a dash (-). If the details of a specific feature is not known (e.g., the details are unavailable) we keep the cell in empty status. In terms of SQL like query languages few EP systems such as TIBCO StreamBase, WSO2 CEP, and SQL Stream supports SQL like query language. Streams Processing Language (SPL) of IBM Infosphere Streams is a data flow language with different syntax from SQL. SPL is a streaming language and it is not a EP language because it does not primarily concentrate on matching patterns over event streams [98]. All except 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 Table 2 we list four types of windows called Sliding windows, Time windows, Tuple windows, and Batch windows. Sliding windows expires only part of the events stored in the window. Time windows completely expires the content of the windows when a predefined elapsed time passes. Tuple window expires its content upon receipt of a punctuation tuple. Batch window (also known as 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 [185] (TIBCO Streambase), [145] (IBM InfosphereStreams), [157] (WSO2 CEP), [176] (SQLStream). While an essential summary of the Event Processing Platforms feature comparison is given in Table 2 extended versions of such comparisons is available in [125][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 geo-distributed 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 for this category of systems [6]. The design of Borealis emphasized three fundamental requirements for stream processing engines

Table 2. Feature comparison of Event Processing Platforms.

Feature	TIBCO Stream-Base	IBM Infosphere Streams	Data Tor- rent RTS	Amazon Kine- sis	WSO2 CEP	Fujitsu Inter- stage CEP	SQL Stream
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/second (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 Geo information 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 Fault Tolerance and High Availability 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 [176][28].

such as (1) Dynamic revision of query results (2) Dynamic query modification (3) Flexible and highly-scalable optimization.

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

Spark-Streaming [200], Samza [21], Apache Flink (previously Stratosphere) [20], Trill [41], Apache Apex [159], etc. are examples for DSCPs which has 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 event processing platforms as such, but can be used to implement event processing functionality.

Geo-distributed 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 event processing systems such as latency, throughput, performance variability, etc. become significant factors in operating a geo-distributed EP installation [191]. Conducting high performance event stream processing across data centers require significant changes from the conventional event processing system architectures. In order to create an efficient event transfer, batched event transferring has to be adapted in such systems. There are two key challenges on event batching, first the size of the event batch and second, 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 multi-route streaming across cloud nodes.

Distributed Stream Computing Platforms are still evolving in various aspects such as declarative query languages, fault tolerance, etc. DSCPs such as Storm has undergone significant architecture enhancements in recent times [189][118]. 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 is equipped with an SQL like query language. All of the DSCPs listed in Table 3 are released under Apache 2.0 license. The key query types 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, while stream joins can be implemented using Apache Samza, its API does not include explicit support for implementing a join operation.

4.3 CEP Libraries

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 event processing 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], Cayuga [34] are some of the notable CEP libraries which appeared recently. A comparison of the features of CEP libraries is shown in Table 4. The dots on each cell indicates the presence of the designated feature on the CEP library. Note that we follow the same convention throughout the rest of the paper.

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	upto 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.

All of the CEP libraries listed on Table 4 are following open source license scheme. Furthermore, none-of them currently support geospatial data types [5]. We conducted an analysis on the technologies used for programming modern open source event processing systems. We have listed the percentages of programming languages used by a selected list of open source event processing 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++, C#.

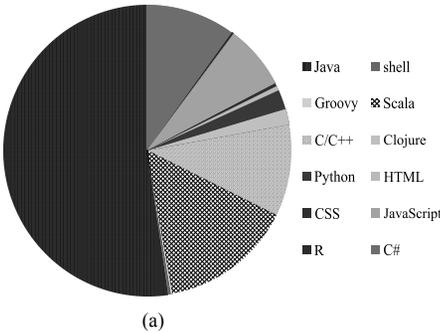
5 EVENT PROCESSING USE CASES

Event Processing paradigm has undergone rapid growth in recent years. Out of a number of topics investigated over the years several areas of research has attracted significant attention. In this section we summarize the most notable recent use cases of event processing.

Table 4. Feature comparison of CEP 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	●	●		
Parameterisation	●	●		
Best reported performance	500 000 event/s [63]	8.55 Million events/s [150]	-	-

Percentage Use of Programming Languages in Total



Percentage Programming Language Usage Per Event Processing System

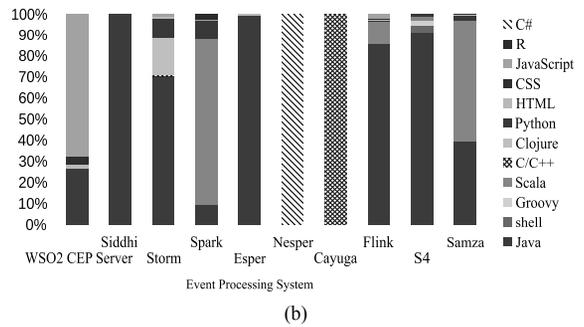


Fig. 2. Event Processing Software Programming Language usage. The pie chart on the left-hand side presents an aggregated view while the right-hand side barchart shows a breakdown of the programming languages used in different event processing systems.

5.1 Internet of Things (IoT) Initiatives

Internet of Things (IoT) has become hot topic in the EP paradigm in recent times. 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 is focused on allowing users to monitor the device’s statistics and to receive alerts about the device. Interesting device level statistics could be obtained from an event processing engine. Analytics conducted across multiple devices allow for aggregating data across different levels of the device hierarchy and let users to post custom queries on the aggregated data. Predictive analytics (e.g., Classification, Predicting the next value, Anomaly detection) enables to select a set of data and conduct advanced analysis using techniques such as machine learning. Applications of EP

technologies can be found on all these three levels. For example, Open Mobile miner can be used to build light-weight mobile applications that run in single device such as Mobile Activity Recognition [72]. Coll-stream is an example for stream processing application which involves multiple devices [82].

There are multiple interesting use cases of EP in IoT related research in recent times. One such example is the Automotive Embedded DSMS (AEDSMS) [199]. The embedded systems installed within a vehicle are used to support autonomous and safe driving. New breed of challenges are faced when designing AEDSMS such as query precompilation, distributed stream processing using in-vehicle networks (network within the vehicle), real-time scheduling and sensor data fusion. General data stream management systems 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 in order avoid accidents. Ability to predict worst case latency is 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 which 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 makes the task of designing an AEDSMS complicated.

Smart city applications are another examples for IoT initiatives. Within buildings there are multiple subsystems such as power generators, heating and ventilations, AC, switch gears, elevators. Smart city applications extract real-time information from such things and run analytics and provide useful information which could be used for saving energy, for providing comfortable environment, for remote maintenance, etc. Smart city applications are hindered by various issues such as difficulty of 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.* presented ACEIS (Automated Complex Event Implementation System) which is an automated discovery and integration system for urban data streams [75]. It receives requirements from users and applications as event request 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 which come in from the connected buildings.

Another example for an IoT event processing scenario can be pointed out from the enterprise communications domain. The work by Ali *et al.* presented a conceptual architecture of IoT-enabled communication systems [15]. IoT applications involving event processing with Smart Grid data have appeared recently [206]. One such use case is the warning of 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 which is closely related to IoT. 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]. Problem is motivated by the size of the IoT data, which can range from Gigabytes to Terabytes per day, and lack of 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 which otherwise would have to be streamed from the event producer to CEP engine. Edge Analytics conducts partial event processing 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.

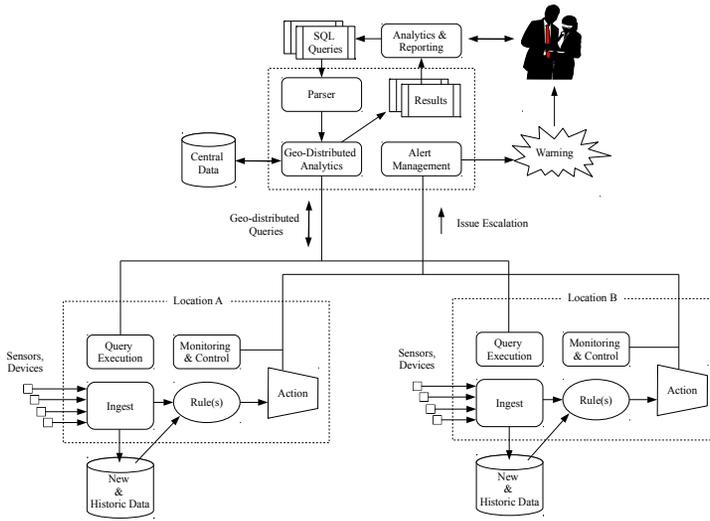


Fig. 3. ParStream's Distributed Architecture for IoT. The federated mode of operation of ParStream is not significantly different from its single cluster operation.

Edge Analytics has significant potential in future EP initiatives since the amount of BigData generated from different streaming sources continue to grow exponentially. ParStream IoT Analytics Platform (shown in Figure 3) is one example for an industrial geo-distributed edge analytics platform [30]. SQL queries are received by ParStream from a variety of analytical applications which get connected to it. ParStream query is broken down into a series of smaller SQL queries for each of the nodes. Each of these sub queries 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 geo-distributed 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 setup distributed execution of EP systems in a geographically distributed area. Some work have been conducted to investigate on collaborations between multiple streaming systems such as CLASP (Collaborating, Autonomous Stream Processing systems) [33]. Such systems operate above desperate data stream processing systems and allows them to cooperate. Chandramouli *et al.* 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 for 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), Online multiplayer games (e.g., Halo : reach on Xbox 360), etc.

Light weight CEP engines have been developed to support edge analytics use cases in recent times. These CEP engines essentially have low resource consumption footprints. One example system is JetStream 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 event processing 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 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 which 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 event processing [66]. Event context and the context-awareness are two very important factors in IoT applications. Three key uses of the context in event processing 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. Event processing agent can be active in some contexts and inactive in others which is called state-oriented context. Akbar *et al.* described a context-aware method to extract and analyze high-level knowledge from data streams [12]. The proposed approach has been implemented on μ CEP which is a lightweight CEP which runs on embedded devices. Similarly Wang *et al.* described a high performance context-aware CEP architecture and method for IoT [195]. They modeled the context as fuzzy ontology which supports linguistic variables and uncertainty in event queries.

5.2 EP on top of Text Data Streams

Complex event processing over text data streams has become a hot topic in recent times due to the increased usage of online social networking sites, news sites, e-commerce, etc. 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 which 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 its customer's behavior and preferences via processing user clicks [130]. Benhardus *et al.* outlined methodologies of detecting trending topics from streaming Twitter data [27].

S³-TM is a stream processing application for streaming short text matching [25]. The work on S³-TM have shown that text matching can be parallelized by using a partitioning of words over matchers. They have shown that co-occurrence relationship between words can make the word partitioning-based routing a scalable and effective solution which provides more than 2.5 times higher throughput compared to a baseline approach.

5.3 EP on top of Video Data Streams

Complex event processing over video streams is an important research direction in recent times due to the prevalence of video streams captured from surveillance cameras located on stationary platforms [44], mobile ground vehicles, Unmanned Air Vehicles (UAVs), etc. [134]. This is very powerful given that camera is one of the most ubiquitous sensing devices available. Several moving object detection systems have been presented in recent times which conduct online analytics on video data streams. Medioni *et al.* presented a technique for analyzing the behavior of the object

movement in a scene [134]. Video streams which are produced by an Unmanned Airborne Vehicle (UAV) are processed to detect and track the object movement information. Certain research conducts event-based video indexing based on their semantical contents. Such work consider multi-modal information streams [23]. Zhang *et al.* modeled the stream processing as a set of task templates whereby each template can be independently instantiated to multiple video streams [202].

Often, video processing is computationally intensive and Field-Programmable Gate Array (FPGAs) have been successfully utilized for accelerating video data stream processing scenarios. Lysakov *et al.* described HDG hardware solution for processing and transmission of TV images, running computation intensive algorithms, mathematical modeling of numerical methods, and automating physical experiments [126].

5.4 Analytics on Top of Graph Data Streams

In the last 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 which represents all the 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. First, novel graph streaming algorithm implementations. Second, streaming graph partitioning which deals with dividing graph data streams into manageable chunks. Third, streaming graph processing on linked data (e.g., RDF data stream processing).

Recently, a number of graph algorithms have been implemented following the data stream processing model. Most of them are streaming implementations of their batched counterparts. One such examples is the streaming implementations of PageRank algorithm. Sarma *et al.* presented a technique for calculating the PageRank of a large graph [52]. Their approach is based on estimating the conductance of the graph. While 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 Sarıyüce *et al.* [164]. Pavan *et al.* presented a technique for triangle counting on a graph stream [149]. Guha *et al.* presented linear sketches for estimating vertex connectivity and constructing hyper graph sparsifiers [87]. 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 recent focus on graph stream processing community. Stanton *et al.* proposed a natural, simple heuristics for streaming graph partitioning and compared their performance to hashing and METIS [180]. They have shown that their heuristics provide significant improvement with 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 DSMS, to enable efficient processing of RDF data using relational model the data needs 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 characteristics of LSD is the primitive operation on RDF stream and

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 optimisation techniques for continuous queries over the triple-based data model.

6 EP OPEN RESEARCH TOPICS

Different aspects of event processing system operation such as event query languages, accuracy, performance tuning, high availability, fault tolerance as well as novel topics such as EP on non-conventional processors, streaming machine learning, etc. has 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 recently. One of the notable advancements is the IBM's Streams Processing Language (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 non-streaming languages (such as C++, 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 event stream processing [175]. Initiatives such as Brooklet calculus are natural abstractions for streaming languages [174]. Use of a calculus enables formal proofs which shows that optimizations made on EP applications are safe and the front-ends realizes the semantics of their source languages properly. Another event processing 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. Use of Rules and use of SQL like query languages are two approaches being followed. Rules enable developers to express application logic in a EP system. Writing rules is a complex and error prone task which needs checking the correctness of rules with respect to the desired application behavior [50]. As a solution Cugola *et al.* have provided a methodology and tool for checking a rule set against application specific properties that have potential of scaling to large sets of mutually interacting rules. Similarly Zhang *et al.* conducted a theoretical results on expressive power and computational complexity of pattern query languages in EP [204].

SQL like queries let end users express complex requirements in a compact form. Furthermore, through operators, they promote reuse of common logic. Moreover, they enable query planning and automatic optimizations. Use of SQL like queries has received wide adoption, and all key open source Event Processing systems such as Apache Strom, Apache Spark Streaming, Apache Fink, WSO2 CEP, SQLStream, etc. supports 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 simple and intuitive manner. Specially such visual interfaces enable non-programmers to write queries easily.

Many real world products (e.g., from IBM, TIBCO, Oracle, Evam, etc.) employ an eclipse based studio for designing, composing queries, and results visually. 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].

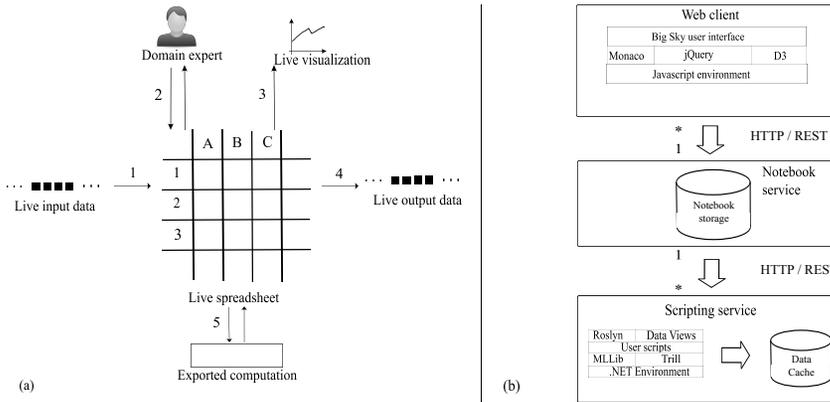


Fig. 4. Different approaches for visualizing data streams. (a) ACTIVESHEETS (b) Tempe

There are two main recent developments in terms of making visualizations of data stream processing: use of spreadsheets and use of notebooks. An example for 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 2D-view of the data, reactive model of computation, and stateless computation present in both the approaches. Their approach proposed a reactive programming model for stream processing which 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 (See Figure 4 (a)). ACTIVESHEETS is based on 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 persisting the computation on the server beyond the life of the spreadsheet.

Data analytics notebooks are gaining huge popularity in last few years as a means of conducting interactive data analysis. However, currently there are very few work being done on 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 which implements the user experience. Second a Notebook Service which stores user-generated content and multiplexes requests from multiple clients. Third, a Scripting Service which provides the computing environment for the user’s script.

6.3 Event Processing with Uncertain Data

Uncertain data streams are data streams which comprise of incomplete and imprecise data [190]. Imprecise events can be caused by factors such as the deficiencies of the measuring accuracy, signal disturbance, privacy protection, etc. 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. On the other hand, an event has existential uncertainty when the sum of the existential probabilities of its possible values is < 1 .

In addressing value uncertainty, cleansing the data is not feasible due to time constraints. Such uncertainties are usually treated with probabilities. Hence it has been argued that event processing engines which could deal with such probabilities be developed [196]. Na *et al.* proposed a model of uncertain complex event processing system for real-time monitoring in product manufacturing process [128]. They defined a 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.* have studied the challenges arising from existential uncertainty [51]. They have extended the semantics of sliding window to define the novel concept of uncertain sliding windows. They provide both exact and approximate algorithms for managing windows under existential uncertainty.

There are few EP systems being developed targeting uncertain event processing. CLARO is a system which supports stream processing for uncertain data [190]. CLARO has a unique data model called mixed-type model where continuous uncertain attributed follow Gaussian mixture distributions. Furthermore, it provides formal semantics, query planing for complex queries, evaluation techniques for relational operators.

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

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 due to multiple reasons such as operator parallelization, network latency, merging of asynchronous streams, etc [108]. Order-less event processing 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 of disorder handling: Buffer-based, Punctuation-based, Speculation-based, and Approximation-based.

Buffer-based techniques. These 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” which 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 T time period and reordering of any out-of-order events which 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 orchestrates returning results pertaining to windows. Hence unlike Buffer-based techniques, out-of-order events can be directly processed by the query.

Speculation-based techniques. These techniques operate as if 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, histograms, etc. They create rough results using the summary information. Table 5 provides a summary of the content present in this subsection.

A main disadvantage of buffer-based disorder handling approaches is sorting and buffering delays the execution of the 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 which signals reception of all delayed arrivals cannot be created. Speculation-based disorder handling will consume 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 considerable number of results with noticeable errors. Overall buffer-based and punctuation-based approaches are conservative approaches while speculation-based and approximation-based approaches are aggressive. The former two techniques delay their processing to gather delayed events so that they could produce results with higher accuracies. The latter two approaches presume there is no disorder in the tuples, continue processing as they are, output the results immediately. They reconciles once a disorder has been detected.

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

6.5 EP Benchmarking and Performance Characterization

EP system performance is a critical quality of service aspect worth of investigation. EP benchmarking itself is a vast area with significant amount of literature. However, EP benchmarking still has number of 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 first we 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 it has been implemented on multiple different CEP systems. Although LR has been introduced circa 2004 it has been widely implemented in multiple CEP systems even in recent ones such as [32]. Among these implementations the LR implementation developed for SCSQ by Zietler *et al.* is of importance [201].

Another notable works of EP benchmarking has been conducted by Mendes *et al.* which is called BiCEP suite [137]. This framework consists of several benchmark specifications (such as Pairs) as well as a workload generator framework called FINCoS. Recently, Nabi *et al.* have implemented a benchmark based on Enron email corpus [145]. CEPsim is a simulator for cloud-based EP systems which 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.

Recently efforts have been made on developing performance models for stream processing systems as a means of formalizing the process of characterizing and predicting the performance of event processing systems. In one such work, Bedini *et al.* presented a set of models that formalize the performance characteristics of a practical stream processing system which follows the Actor Model [26]. They model the characteristics of the data flow 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.

Table 5. A summary of the out-of-order event processing techniques.

Approach	Technique	Reference(s)	Characteristics	Limitations
K-slack	Buffer-based	[144]	Transparently buffers and reorders 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	[108]	Dynamically adjusts the size of the input buffer to minimize the result latency.	Limited to sliding window aggregates.
Tirthapura <i>et al.</i>	Approximation-based	[45][188]	Aggregates over polynomial and general decay functions.	Produces significant amount of errors for queries with small windows
Brito <i>et al.</i>	Speculation-based	[35][123]	These techniques are vigorous. They presume no disorder present and create results at the same time when window closes. 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 final revision. This will consume lot of CPU and will introduce large latency in the results.
Krishnamurthy <i>et al.</i>	Punctuation-based	[115]	Depends on punctuations within data streams. Punctuations explicitly triggers the operator to send results.	Latency problem of buffer-based techniques still exists with this approach.

Chauhan *et al.* conducted an empirical evaluation of 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 by 11 [43]. Similarly Xu *et al.* conducted performance study of T-Storm stream processing system with three benchmarks called Throughput Test, Word Count, and Log Stream Processing [198]. Zubair *et al.* [145] conducted a detailed study of Storm's performance. Several works such as [58][136] have conducted workload characterizations across different EP systems. Factors affecting a virtualized event processing system were analyzed by Ku *et al.* [117].

There have been recent studies conducted considering other novel dimensions of performance of stream processing software such as [55][182]. However, they characterize the energy consumption of stream processing systems. Furthermore, certain CEP benchmarking works have considered the scenarios having 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].

Table 6. A summary of the EP benchmarks.

Benchmark	Reference(s)	Characteristics	Limitations
LinearRoad	[22]	Application.	Complex to implement, difficult to parallelize
FINCoS	[137]	A set of benchmarking tools for load generation and performance measurement.	Limited scalability.
Email Processor	[145][157]	Application.	Does not include EP operations such as windows.
CityBench	[14]	RDF Stream Processing benchmark.	Limited scalability.

DEBS Grand Challenge is a notable initiative made by the event processing community to characterize and improve the performance of event processing systems. DEBS Grand Challenge was introduced to DEBS conference series in year 2011 by Opher Etzion. Each year the competition introduces a unique real world application scenario of event processing and ranks the solutions mainly based on the performance (e.g., throughput and latency) of the solution. Since its inception, DEBS Grand Challenge has attracted significant attention from the event processing community. In the most recent version of the challenge [106] (DEBS 2015) 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 which is developed for evaluating RDF stream processing (RSP) engines 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 stream processing engines were evaluated using the CityBench to evaluate their capabilities and limitations to operate as RSPs in smart city applications.

6.6 EP Provenance

Provenance is a detailed record of how an output data element is produced by a system, including detail about inputs it received. It is used to describe the incoming data and the intermediary state which 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 [81][80]. EP provenance helps human supervisors to diagnose how certain events were triggered in the system and react accordingly.

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

High stream event rate and low latency are the two key performance requirements with most event processing applications. Hence when enabling the background processes such as provenance it is important that such processes do not degrade these 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 which 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 Event Processing [91]. Multiple work has been conducted in conjunction with EP in cloud environments in recent times. Amazon Kinesis Streams is an example effort made by the industry [16]. Clouds have their own inherent issues. The key issues being fault tolerance, 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 fault tolerance in cloud computing systems in Section 6.8.

Elastic load balancing is essential to operate between multiple different clouds. Kleiminger *et al.* presented a combined stream processing system which adaptively balances workload between two stream processors deployed in a private cloud as well as in a public cloud [113]. They implemented and evaluated their approach with 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.* on the other hand mentioned that the main factor dominating performance of elastic load balancing with clouds is the latency introduced by the cloud deployment [37]. Ishii *et al.* have created a method and an architecture which transfers data stream processing to a cloud environment as required by monitoring the data rate of the input data stream [104]. Heinze *et al.* introduced an elastic data stream processing system which optimizes its utilization under certain latency constraints defined by a SLA [93]. It was developed as an online parameter optimization.

Privacy and security is of utmost importance in the context of cloud computing. Multiple techniques have been exploited for implementing privacy preserving stream data analytics which can be effectively used for securing the streaming data in clouds. Out of number of algorithmic techniques, randomized method has been used in this regard. Randomization method adds noise to the streaming data in-order 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.* described mechanism for collecting privacy sensitive data from users and calculate aggregate statistics periodically [39]. They introduces a new technique to achieve fault tolerance while incurring very small penalty of communication overhead and estimation error.

6.8 High Availability and Fault Tolerance

High Availability (HA) and Fault Tolerance (FT) are closely related yet two different terms which are often get confused. In this section first we differentiate between these two terms. Next, we describe how HA has been implemented in event processing systems and then move onto describing different approaches which 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. High availability (HA) is critical for operating a CEP system smoothly. High availability enables system to be accessible 24×7. However, high availability 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]. Fault tolerance (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.* described a hybrid HA method which combines the benefits of both AS and PS approaches [205].

The system operates in PS mode during normal conditions and quickly switches to the AS mode by activating a predeployed secondary copy which 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 operation of an EP system are defined in terms of delivery guarantees found in messaging systems: *At Least Once*, *At Most Once*, and *Exactly Once*. Recent event processing software such as Storm, MillWheel [13], Spark Streaming [200], Apache Flink, 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 on the other hand follows "At Least Once" guarantee which results in low throughput. A detailed comparison of these fault tolerance models is available in [192]. Flink is a good example of a project that has "Exactly Once" where it creates consistent checkpoints of event processing graph using variation of global snapshot algorithm and restart the job from checkpoints in case of a failure. *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 CEP community which is important for fault tolerant operation of EP systems. Several event processing systems have appeared to address the issue of introducing transactional guarantees to data stream processing. S-Store is an example for such 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 which follows *transactional updates* model [192]. In such model each intermediate record which passes through an operator (together with the derived records and state updates) creates a commit record which 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 as well as replay of the lost records is also carried out. However, 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 having 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 comprises of a mechanism which tracks fine-grained data dependencies between the output and the input streams to enable efficient recomputation-based failure recovery which achieves strong exactly-once semantics.

EP systems often have to balance the two QoS parameters latency and throughput. Lohrmann *et al.* described an approach which 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 have implemented and evaluated their approach. Their approach improved the workflow latency by at least fifteen times with 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 using multiple machines (horizontal scaling).

In the era of multicore computing, EP software systems need to utilize all the available cores of a computer system in an efficient manner. EP optimizations can be categorized under two main areas as Blackbox optimizations (E.g., Distribution, Parallelism, Scheduling, Load balancing, Load shedding, etc.) and Whitebox optimizations (e.g., Implementation selection, Implementation optimization, Pattern rewriting, etc.) [67]. In EP systems parallelism can be achieved through one of the three techniques: data parallelism, task parallelism, and pipeline parallelism [165].

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

Auto-parallelization of data stream processing systems is another area that is widely being studied. A number of auto parallelization work have been conducted in the context of IBM Infosphere Streams (i.e., System S). The work by Dayarathna *et al.* described data flow graph transformation as a technique for auto tuning a streaming application run in System S [56][57]. Schneider *et al.* and Gedik *et al.* [77] addressed the issue of auto tuning with the presence of stateful operators [165].

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

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

Due to the highly dynamic nature of input data stream, identifying a single plan results in a suboptimal query plan. Multi-route approach which is a new stream processing technique has appeared to solve this issue [36]. Cao *et al.* described one of the first practical multi-route optimizer called Correlation-aware Multi-route Stream Query Optimizer (CMR) which solves the above issue. In this paper we describe the most commonly encountered scaling techniques. It can be observed that system scalability experiments conducted till present are focused on full system performance. However, system performance characteristics in special scenarios such as performance behavior during system failures, data partitioning, windowing, etc. needs to be addressed in detail. Summary of these techniques is listed in 7.

6.10 Approximate complex event processing

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.

Table 7. Comparison of event processing scaling approaches.

Approach	Reference(s)	Advantages	Limitations	Comments
Fission (i.e., Parallelization)	[57][77]	Leverage multi-cores	Difficult to partition shared state	Dynamic determination of level of parallelism increases the performance portability
Fusion	[111][57][77]	Reduced complexity of accessing shared state	Cannot leverage multicores	Can be done either following top-down or bottom-up approaches
Pipelining	[83]	Leverage multi-cores	Adds additional path to the flow graph per each pipeline stage created	Pipelining may enable other optimizations such as operator reordering
Rewriting	[167][154]	Improved performance on the same hardware environment	Application specific	Suited for applications which involve complex patterns detection
Operator reordering	[152]	Improved performance on the same hardware environment	Preconditions for safe operator graph changes are very restrictive	Fruitful if it moves selective operators before costly operators

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

Rusu *et al.* 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 [161]. Garofalakis *et al.* proposed a novel algorithmic technique for efficiently tracking sketch-based approximations for a broad class of complex aggregate queries over massive, distributed data streams [76]. The tracking protocols 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 AMS (Alon, Matias, Szegedy) 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 *et al.* discussed use of Sketching techniques for historical diagnosis of sensor networks [10]. They specifically discussed methods which are accurate enough to reconstruct any portion of the historical sensor signal with a relatively small amount of pre-stored 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 Non-conventional Processors (GPUs and FPGAs)

Advancement in use of non-conventional high performance device technologies such as GPUs and FPGAs provide an opportunity build very fast event based systems. High performance, massively parallel accelerators such as GPUs, FPGAs, etc. 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 which 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 stream processing engine where an optimizer distributes query workloads across a set of CPUs and FPGA chips. Mueller *et al.*'s work have 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, the actual performance which can be reached with the resulting system. They have found that it is really difficult to generate really high packet rates in lab settings to stress the FPGA. Using a NetBSD-based packet generator, the authors generated a maximum of 1,000,400 packets/s on UDP. They have shown that FPGA can process streams at network speed where the bottleneck exists with the network.

It is difficult to use a multi-core processor like Cell for event processing mainly due to the heterogeneous nature of the processing elements. The co-processor side local memory and the unconventional programming model usually inhibits their wide-spread adoption [78]. In search of a solution Gedik *et al.* studied about execution of windowed stream join operators in a scalable manner on multi-core processors (specifically targeting Cell processors). Their approach achieves $8.3\times$ higher output rate compared to a naive approach on two 3.2 GHz Intel Xeon processors.

Cugola *et al.* have investigated how parallel hardware may speed up CEP [47]. They considered the most common operators offered by the current rule languages (e.g., sequences, parameters, aggregates, etc.) and considered different algorithms to process rules built using such operators. They discussed how such algorithms can be implemented on multi-core CPU and on CUDA. Verner *et al.* have designed a framework for hard real-time stateful processing of multiple streams on heterogeneous platforms with multiple CPUs and a single accelerator [193]. Margara *et al.* has shown that use of parallel hardware can provide impressive speedups in content-based matching. They have developed Parallel Content Matching (PCM) algorithm to leverage parallel hardware to perform publish-subscribe content-based matching [129]. Giakoumakis *et al.* describe 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 [79]. In particular they mapped Very Fast Decision Tree (VFDT) streaming algorithm on a single custom processor in a high-end Convey HC-2ex FPGA platform. Karnagel *et al.* proposed a variant of a window-based stream join operator that is optimized for processors with heterogeneous execution units [110]. The join operation is performed within windows of the stream and get scheduled as a task within a queue. Furthermore, they have proposed HELLS-Join approach which uses all heterogeneous devices by segregating parts of the algorithm on to appropriate device [109]. They have shown that HELLS-Join performs better than CPU stream joins which allows for using wider time windows, higher stream frequencies, and more streams to be joined.

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

6.12 Streaming Processing with DBMSs

Event processing has its roots in the 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 there have been increasing importance of investigating the usage of RDBMSs for data stream processing. On the other hand there are few works 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 recent work done in this area.

Liarou *et al.* have designed a stream engine on top of an existing relational database kernel. This has 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 database management system (DBMS) at the query plan level [122]. Joining with data in an RDBMS is conducted via windows where every time a window is complete the result over all the tuples in that window is calculated.

Feasibility of using database systems to assist stream processing engines (SPE) to process complex aggregate queries to reduce their evaluation latency has been an area investigated recently [107]. Dynamic migration of complex aggregate operations between the SPE and the database engine in response to varying system load have been investigated by Ji *et al.*

6.13 Streaming Machine Learning

Machine Learning (ML) learns an underline function given sample data [1]. Most of the ML algorithms are initially developed targeting batched data analytics. Lot of machine learning 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 data sets in the Internet companies have resulted in a necessity of conducting ML on streaming data. Several libraries have appeared to solve the problem of Streaming ML recently.

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

Stream clustering has been widely studied algorithm in recent times. 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 have 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 which can handle such changes in concepts over time. Bifet *et al.* 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 [29].

Table 8. Comparison of streaming ML projects. Note that all of these are open source ML projects.

Project	Example Real-world Applications	Distributed?	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++	-	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	-	-

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

7 CONCLUSIONS

Event processing has established as a paradigm for data processing for nearly three decades. We have observed significant advancements made for EP during the last 5 to 10 years period. This paper created a conscious summary of these research by classifying the entire body of research into three categories as EP system architectures, EP use cases, and open research topics. It can be observed that significant work carried out in the EP system architectures especially focusing on the DSCPs. Among the notable EP use cases EP on IoT, EP on Text Data Streams, Data stream processing in clouds remain as 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 event processing languages, and use of heterogeneous devices for event processing (see Figure 5).

While most of the modern open source event processing 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 event processing systems. These trends are reflected from 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 still remains in their early stages. There are significant recent efforts made on IoT related EP. Graph streaming processing is another interesting area which is still at its infancy. We envision promising research advances in the areas of IoT, graph streams, and Streaming ML in near future.

Open research topics in CEP	Query languages and visual interfaces						2		1	1	1	4
	Uncertainty								1	1		2
	Non-conventional devices					2		1	1	2	2	1
	Approximate CEP		1	1	1				1	1	1	2
	System scalability and Auto tuning						1	2	2	4	4	
	HA and Fault Tolerance	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 Machine Learning					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 on EP. The timeline provides a summary of the main topics surveyed in this paper.

Significant amount of work needs to be done in the EP benchmarking aspects. While number of 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 while none of the existing benchmarks address the scenarios of graph stream processing. Furthermore, there is no concession among the researchers in EP on a standard benchmark for system performance.

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

While Streaming ML has attracted significant attention recently not many real world implementations of the streaming ML algorithms are available currently. Furthermore, such algorithms' performance behaviors are still remain as unknown.

In summary the above analysis indicates that there has been significant growth in event processing applications and the related technologies in the last 5 to 10 years time. We expect a similar growth of the event processing industry further in the next decade as highlighted by our analysis.

REFERENCES

- [1] 2012. Mathematical Models. In *Encyclopedia of the Sciences of Learning*, NorbertM. Seel (Ed.). Springer US, 2113–2113.
- [2] 2014. *Complex Event Processing (CEP) Market - Global forecast to 2019*. RESEARCH AND MARKETS.
- [3] 2015. Dell Edge Gateway 5000 Series. URL: <http://i.dell.com/sites/doccontent/corporate/secure/en/Documents/edge-gateway-specsheet.pdf>, (2015).
- [4] 2015. *Streaming Analytics Market by Verticals - Worldwide Market Forecast & 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. (2014).

- [6] Daniel J Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Cetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag Maskey, Alex Rasin, Esther Ryzkina, and others. 2005. The Design of the Borealis Stream Processing Engine.. In *CIDR*, Vol. 5. 277–289.
- [7] Raman Adaikkalavan and Sharma Chakravarthy. 2006. SnoopIB: Interval-based event specification and detection for active databases. *Data & Knowledge Engineering* 59, 1 (2006), 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 Comput. Surv.* 47, 1, Article 10 (May 2014), 36 pages.
- [9] CharuC. Aggarwal and PhilipS. Yu. 2008. A General Survey of Privacy-Preserving Data Mining Models and Algorithms. In *Privacy-Preserving Data Mining*, CharuC. Aggarwal and PhilipS. 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 *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*. 185–194.
- [11] Charu C. Aggarwal. 2013. A Survey of Stream Clustering Algorithms. In *Data Clustering: Algorithms and Applications*. 231–258.
- [12] A. Akbar, F. Carrez, K. Moessner, J. Sancho, and J. Rico. 2015. Context-aware stream processing for distributed IoT applications. In *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*. 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. *Proc. VLDB Endow.* 6, 11 (Aug. 2013), 1033–1044.
- [14] Muhammad Intizar Ali, Feng Gao, and Alessandra Mileo. 2015a. CityBench: A Configurable Benchmark to Evaluate RSP Engines Using Smart City Datasets. In *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part II*. 374–389.
- [15] Muhammad Intizar Ali, Naomi Ono, Mahedi Kaysar, Keith Griffin, and Alessandra Mileo. 2015b. A Semantic Processing Framework for IoT-Enabled Communication Systems. In *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part II*. 241–258.
- [16] Inc. Amazon Web Services. 2016. Amazon Kinesis Streams. URL: <https://aws.amazon.com/kinesis/streams>. (October 2016).
- [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 (2014), 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, USA, 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 *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*. 137–148.
- [20] Apache Software Foundation. 2015a. Apache Flink: Scalable Batch and Stream Data Processing. URL: <https://flink.apache.org/>. (September 2015).
- [21] Apache Software Foundation. 2015b. Samza. URL: <http://samza.apache.org/>. (July 2015).
- [22] Arvind Arasu, Mitch Cherniack, Eduardo Galvez, David Maier, Anurag S. Maskey, Esther Ryzkina, Michael Stonebraker, and Richard Tibbetts. 2004. Linear Road: A Stream Data Management Benchmark. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30 (VLDB '04)*. VLDB Endowment, 480–491.
- [23] N. Babaguchi, Y. Kawai, and T. Kitahashi. 2002. Event based indexing of broadcasted sports video by intermodal collaboration. *Multimedia, IEEE Transactions on* 4, 1 (Mar 2002), 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 Basik, Buğra Gedik, Hakan Ferhatosmanoğlu, and MertEmin Kalender. 2015. S3-TM: scalable streaming short text matching. *The VLDB Journal* 24, 6 (2015), 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, USA, 173–184.
- [27] James Benhardus and Jugal Kalita. 2013. Streaming Trend Detection in Twitter. *Int. J. Web Based Communities* 9, 1 (Jan. 2013), 122–139.
- [28] Rahul Bhartia. 2014. Amazon Kinesis and Apache Storm. (2014).

- [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)*. Springer-Verlag, Berlin, Heidelberg, 249–260.
- [30] Robin Bloor and Rebecca Jozwiak. 2014. A Database Platform for the Internet of Things. (2014).
- [31] M. Blount, M.R. Ebling, J.M. Eklund, A.G. James, C. McGregor, N. Percival, K.P. Smith, and Daby Sow. 2010. Real-Time Analysis for Intensive Care: Development and Deployment of the Artemis Analytic System. *Engineering in Medicine and Biology Magazine, IEEE* 29, 2 (March 2010), 110–118.
- [32] I. Botan, Younggoo Cho, R. Derakhshan, N. Dindar, A. Gupta, L. Haas, Kihong Kim, Chulwon Lee, G. Mundada, Ming-Chien Shan, N. Tatbul, Ying Yan, Beomjin Yun, and Jin Zhang. 2010. A demonstration of the MaxStream federated stream processing system. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*. 1093–1096.
- [33] Michael Branson, Fred Douglass, Brad Fawcett, Zhen Liu, Anton Riabov, and Fan Ye. 2007. CLASP: Collaborating, Autonomous Stream Processing Systems. In *Middleware 2007*, Renato Cerqueira and RoyH. Campbell (Eds.). Lecture Notes in Computer Science, Vol. 4834. Springer Berlin Heidelberg, 348–367.
- [34] Lars Brenna, Alan Demers, Johannes Gehrke, Mingsheng Hong, Joel Oshser, 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, USA, 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 Second International Conference on Distributed Event-based Systems (DEBS '08)*. ACM, New York, NY, USA, 265–275.
- [36] Lei Cao and Elke A. Rundensteiner. 2013. High Performance Stream Query Processing with Correlation-aware Partitioning. *Proc. VLDB Endow.* 7, 4 (Dec. 2013), 265–276.
- [37] J. Cervino, E. Kalyvianaki, J. Salvachua, and P. Pietzuch. 2012. Adaptive Provisioning of Stream Processing Systems in the Cloud. In *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on*. 295–301.
- [38] Ugur Cetintemel, Jiang Du, Tim Kraska, Samuel Madden, David Maier, John Meehan, Andrew Pavlo, Michael Stonebraker, Erik Sutherland, Nesime Tatbul, Kristin Tufte, Hao Wang, and Stanley Zdonik. 2014. S-Store: A Streaming NewSQL System for Big Velocity Applications. *Proc. VLDB Endow.* 7, 13 (Aug. 2014), 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*, AngelosD. Keromytis (Ed.). Lecture Notes in Computer Science, Vol. 7397. Springer Berlin Heidelberg, 200–214.
- [40] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly Detection: A Survey. *ACM Comput. Surv.* 41, 3, Article 15 (July 2009), 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. *Proc. VLDB Endow.* 8, 4 (Dec. 2014), 401–412.
- [42] Badrish Chandramouli, Suman Nath, and Wenchao Zhou. 2013. Supporting Distributed Feed-following Apps over Edge Devices. *Proc. VLDB Endow.* 6, 13 (Aug. 2013), 1570–1581.
- [43] J. Chauhan, S.A. Chowdhury, and D. Makaroff. 2012. Performance Evaluation of Yahoo! S4: A First Look. In *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2012 Seventh International Conference on*. 58–65.
- [44] Inc. Cisco Systems. 2015. Cisco Video Surveillance Stream Manager Software. (2015).
- [45] Graham Cormode, Flip Korn, and Srikanta Tirthapura. 2008. Time-decaying Aggregates in Out-of-order Streams. In *Proceedings of the Twenty-seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '08)*. ACM, New York, NY, USA, 89–98.
- [46] Gianpaolo Cugola and Alessandro Margara. 2010. TESLA: A Formally Defined Event Specification Language. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems (DEBS '10)*. ACM, New York, NY, USA, 50–61.
- [47] Gianpaolo Cugola and Alessandro Margara. 2012a. Low latency complex event processing on parallel hardware. *J. Parallel and Distrib. Comput.* 72, 2 (2012), 205 – 218.
- [48] Gianpaolo Cugola and Alessandro Margara. 2012b. Processing Flows of Information: From Data Stream to Complex Event Processing. *ACM Comput. Surv.* 44, 3, Article 15 (June 2012), 62 pages.
- [49] Gianpaolo Cugola and Alessandro Margara. 2013. Deployment strategies for distributed complex eventprocessing. *Computing* 95, 2 (2013), 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, USA, 10–21.
- [51] Michele Dallachiesa, Gabriela Jacques-Silva, Bura Gedik, Kun-Lung Wu, and Themis Palpanas. 2015. Sliding windows over uncertain data streams. *Knowledge and Information Systems* 45, 1 (2015), 159–190.

- [52] Atish Das Sarma, Sreenivas Gollapudi, and Rina Panigrahy. 2008. Estimating PageRank on Graph Streams. In *Proceedings of the Twenty-seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '08)*. ACM, New York, NY, USA, 69–78.
- [53] DataTorrent. 2015. DataTorrent RTS - DataTorrent. URL: <https://www.datatorrent.com/product/datatorrent-rtls/>. (August 2015).
- [54] DataTorrent. 2016. DataTorrent RTS: Real-Time Streaming Analytics for Big Data. (2016).
- [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 (2017), 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, USA, 335–346.
- [57] Miyuru Dayarathna and Toyotaro Suzumura. 2013a. Automatic Optimization of Stream Programs via Source Program Operator Graph Transformations. *Distrib. Parallel Databases* 31, 4 (Dec. 2013), 543–599.
- [58] Miyuru Dayarathna and Toyotaro Suzumura. 2013b. A Performance Analysis of System S, S4, and Esper via Two Level Benchmarking. In *Quantitative Evaluation of Systems*. Lecture Notes in Computer Science, Vol. 8054. Springer Berlin Heidelberg, 225–240.
- [59] Gianmarco De Francisci Morales and Albert Bifet. 2015. SAMOA: Scalable Advanced Massive Online Analysis. *J. Mach. Learn. Res.* 16, 1 (Jan. 2015), 149–153.
- [60] Xenofontas Dimitropoulos, Marc Stoecklin, Paul Hurley, and Andreas Kind. 2008. The Eternal Sunshine of the Sketch Data Structure. *Comput. Netw.* 52, 17 (Dec. 2008), 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 (2014), 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 Computer Society, Washington, DC, USA, 466–475.
- [63] EsperTech. 2016. Performance Results. URL: <http://www.espertech.com/esper/release-5.3.0/esper-reference/html/performance.html#performance-results>. (2016).
- [64] EsperTech. 2015. Esper: Event Processing for Java. URL: <http://www.espertech.com/products/esper.php>. (August 2015).
- [65] Opher Etzion. 2009. Complex Event. In *Encyclopedia of Database Systems*, LING LIU and M.TAMER ZSU (Eds.). Springer US, 411–412.
- [66] Opher Etzion, Yonit Magid, Ella Rabinovich, Inna Skarbovsky, and Nir Zolotorevsky. 2011a. *Context-Based Event Processing Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, 257–278. DOI: http://dx.doi.org/10.1007/978-3-642-19724-6_12
- [67] Opher Etzion, Ella Rabinovich, and Inna Skarbovsky. 2011b. 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, USA, 365–366.
- [68] Tao Feng. 2015. Benchmarking Apache Samza: 1.2 million messages per second on a single node. URL: <https://engineering.linkedin.com/performance/benchmarking-apache-samza-12-million-messages-second-single-node>. (August 2015).
- [69] Ioannis Flouris, Nikos Giatrakos, Minos Garofalakis, and Antonios Deligiannakis. 2015. Issues in Complex Event Processing Systems. In *In Proceedings of the 1st IEEE International Workshop on Real Time Data Stream Analytics (RTStreams'15)*. IEEE, 6.
- [70] Fujitsu. 2016. FUJITSU Software Interstage Big Data Complex Event Processing Server - Benefits. URL: <http://www.fujitsu.com/global/products/software/middleware/application-infrastructure/interstage/solutions/big-data/bdcep/benefits/>. (October 2016).
- [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 Fifth Balkan Conference in Informatics*. Citeseer, 26–31.
- [72] Mohamed Medhat Gaber, João Gama, Shonali Krishnaswamy, João Bártoolo Gomes, and Frederic Stahl. 2014. Data Stream Mining in Ubiquitous Environments: State-of-the-art and Current Directions. *Wiley Int. Rev. Data Min. and Knowl. Disc.* 4, 2 (March 2014), 116–138.
- [73] Joo Gama, Raquel Sebastiao, and PedroPereira Rodrigues. 2013. On evaluating stream learning algorithms. *Machine Learning* 90, 3 (2013), 317–346.
- [74] João Gama, Indrè Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A Survey on Concept Drift Adaptation. *ACM Comput. Surv.* 46, 4, Article 44 (March 2014), 37 pages.

- [75] Feng Gao, Muhammad Intizar Ali, and Alessandra Mileo. 2014. Semantic Discovery and Integration of Urban Data Streams. In *Proceedings of the Fifth Workshop on Semantics for Smarter Cities a Workshop at the 13th International Semantic Web Conference (ISWC 2014)*, Riva del Garda, Italy, October 19, 2014. 15–30.
- [76] Minos Garofalakis, Daniel Keren, and Vasilis Samoladas. 2013. Sketch-based Geometric Monitoring of Distributed Stream Queries. *Proc. VLDB Endow.* 6, 10 (Aug. 2013), 937–948.
- [77] Buğra Gedik. 2014. Partitioning Functions for Stateful Data Parallelism in Stream Processing. *The VLDB Journal* 23, 4 (Aug. 2014), 517–539.
- [78] Buğra Gedik, Rajesh R. Bordawekar, and Philip S. Yu. 2009. CellJoin: A Parallel Stream Join Operator for the Cell Processor. *The VLDB Journal* 18, 2 (April 2009), 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*, Kentaro Sano, Dimitrios Soudris, Michael Hbner, and Pedro C. Diniz (Eds.). Lecture Notes in Computer Science, Vol. 9040. Springer International Publishing, 357–364.
- [80] Boris Glavic, Kyumars Sheykh Esmaili, Peter M. Fischer, and Nesime Tatbul. 2014. Efficient Stream Provenance via Operator Instrumentation. *ACM Trans. Internet Technol.* 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, USA, 39–50.
- [82] JOO BRTOLO 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 & Decision Making* 12, 06 (2013), 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*, Raghunath Nambiar and Meikel Poess (Eds.). Lecture Notes in Computer Science, Vol. 7144. Springer Berlin Heidelberg, 83–96.
- [85] Jamie Grier. 2016. Extending the Yahoo! Streaming Benchmark. URL: <http://data-artisans.com/extending-the-yahoo-streaming-benchmark/>. (Feb 2016).
- [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)*. Springer-Verlag New York, Inc., New York, NY, USA, 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, USA, 241–247.
- [88] V. Gulisano, R. Jimenez-Peris, M. Patino-Martinez, and P. Valduriez. 2010. StreamCloud: A Large Scale Data Streaming System. In *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*. 126–137.
- [89] Jagabondu Hazra, Kaushik Das, Deva P. Seetharam, and Amith Singhee. 2011. Stream Computing Based Synchronphasor Application for Power Grids. In *Proceedings of the First International Workshop on High Performance Computing, Networking and Analytics for the Power Grid (HiPCNA-PG '11)*. ACM, New York, NY, USA, 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, USA, 63–74.
- [91] Thomas Heinze, Leonardo Aniello, Leonardo Querzoni, and Zbigniew Jerzak. 2014a. Cloud-based Data Stream Processing. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems (DEBS '14)*. ACM, New York, NY, USA, 238–245.
- [92] Thomas Heinze, Valerio Pappalardo, Zbigniew Jerzak, and Christof Fetzer. 2014b. 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, USA, 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 Sixth ACM Symposium on Cloud Computing (SoCC '15)*. ACM, New York, NY, USA, 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 *Big Data (BigData Congress), 2015 IEEE International Congress on*. 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 Second International Conference on Distributed Event-based Systems (DEBS*

- '08). ACM, New York, NY, USA, 95–102.
- [96] Annika Hinze, Kai Sachs, and Alejandro Buchmann. 2009. Event-based Applications and Enabling Technologies. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems (DEBS '09)*. ACM, New York, NY, USA, 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, USA, 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 (May 2013), 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 Second Workshop on Software Engineering Methods in Spreadsheets co-located with the 37th International Conference on Software Engineering (ICSE 2015)*, Florence, Italy, May 18, 2015. 39–40.
- [100] Martin Hirzel, Robert Soulé, Scott Schneider, Buğra Gedik, and Robert Grimm. 2014. A Catalog of Stream Processing Optimizations. *ACM Comput. Surv.* 46, 4, Article 46 (March 2014), 34 pages.
- [101] T.Ryan Hoens, Robi Polikar, and NiteshV. Chawla. 2012. Learning from streaming data with concept drift and imbalance: an overview. *Progress in Artificial Intelligence* 1, 1 (2012), 89–101.
- [102] Geoff Hulten and Pedro Domingos. 2003. VFML – A toolkit for mining high-speed time-changing data streams. (2003). <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 Computer Society, Washington, DC, USA, 779–790.
- [104] A. Ishii and T. Suzumura. 2011. Elastic Stream Computing with Clouds. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. 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, USA, 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, USA, 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, USA, 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, USA, 68–79.
- [109] Tomas Karnagel, Dirk Habich, Benjamin Schlegel, and Wolfgang Lehner. 2013a. The HELLS-join: A Heterogeneous Stream Join for Extremely Large Windows. In *Proceedings of the Ninth International Workshop on Data Management on New Hardware (DaMoN '13)*. ACM, New York, NY, USA, Article 2, 7 pages.
- [110] Tomas Karnagel, Benjamin Schlegel, Dirk Habich, and Wolfgang Lehner. 2013b. Stream Join Processing on Heterogeneous Processors.. In *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)*. Springer-Verlag New York, Inc., New York, NY, USA, Article 16, 20 pages.
- [112] Arush Kharbanda. 2015. Fault Tolerant Stream Processing with Spark Streaming. URL: <https://www.sigmoid.com/fault-tolerant-stream-processing/>. (April 2015).
- [113] W. Kleiminger, E. Kalyvianaki, and P. Pietzuch. 2011. Balancing load in stream processing with the cloud. In *Data Engineering Workshops (ICDEW), 2011 IEEE 27th International Conference on*. 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 Tenth ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS '14)*. ACM, New York, NY, USA, 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, USA, 1081–1092.
- [116] S.P.T. Krishnan and JoseL.Ugia Gonzalez. 2015. Google Cloud Dataflow. In *Building Your Next Big Thing with Google Cloud Platform*. Apress, 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 (2014), 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, USA, 239–250.
- [119] Kx Systems, Inc. 2016. Real time analytics , Kx capabilities , Kx Systems. URL: <https://kx.com/real-time-in-memory-analytics.php>. (January 2016).
- [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 - Volume Part II (ISWC'12)*. Springer-Verlag, Berlin, Heidelberg, 300–312.
- [121] Boduo Li, Yanlei Diao, and Prashant Shenoy. 2015. Supporting Scalable Analytics with Latency Constraints. *Proc. VLDB Endow.* 8, 11 (July 2015), 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, USA, 501–512.
- [123] Mo Liu, Ming Li, D. Golovnya, E.A. Rundensteiner, and K. Claypool. 2009. Sequence Pattern Query Processing over Out-of-Order Event Streams. In *Data Engineering, 2009. ICDE '09. IEEE 25th International Conference on.* 784–795.
- [124] Björn Lohrmann, Daniel Warneke, and Odej Kao. 2012. Massively-parallel Stream Processing Under QoS Constraints with Nephelê. In *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing (HPDC '12)*. ACM, New York, NY, USA, 271–282.
- [125] David Luckham. 2016. Proliferation of Open Source Technology for Event Processing. URL: <http://www.complexevents.com/2016/06/15/proliferation-of-open-source-technology-for-event-processing/>. (June 2016).
- [126] K.F. Lysakov and M.Yu. Shadrin. 2013. FPGA-based hardware accelerator for high-performance data-stream processing. *Pattern Recognition and Image Analysis* 23, 1 (2013), 26–34.
- [127] MahmoudS. 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*. Qing Liu, Quan Bai, Stephen Giugni, Darrell Williamson, and John Taylor (Eds.). Studies in Computational Intelligence, Vol. 426. 129–156.
- [128] Na Mao and Jie Tan. 2015. Complex Event Processing on uncertain data streams in product manufacturing process. In *Advanced Mechatronic Systems (ICAMechS), 2015 International Conference on.* 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 (2014), 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, USA, 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, USA, 269–276.
- [132] Henry May, David Engebretsen, and Walt Madden. 2016. IBM InfoSphere Streams v4.0 Performance Best Practices. URL: https://developer.ibm.com/streamsdev/wp-content/uploads/sites/15/2015/04/Streams_4.0.0_0_PerformanceBestPractices.pdf, (2016).
- [133] Andrew McGregor. 2014. Graph Stream Algorithms: A Survey. *SIGMOD Rec.* 43, 1 (May 2014), 9–20.
- [134] Gérard 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 (2001), 873–889.
- [135] John Meehan, Nesime Tatbul, Stanley B. Zdonik, Cansu Aslantas, Ugur Çetintemel, Jiang Du, Tim Kraska, Samuel Madden, David Maier, Andrew Pavlo, Michael Stonebraker, Kristin Tufté, and Hao Wang. 2015. S-Store: Streaming Meets Transaction Processing. *CoRR* abs/1503.01143 (2015).
- [136] Marcelo R. Mendes, Pedro Bizarro, and Paulo Marques. 2009. Performance Evaluation and Benchmarking. Springer-Verlag, Berlin, Heidelberg, Chapter A Performance Study of Event Processing Systems, 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, USA, 431–432.
- [138] Xiangrui Meng, Joseph K. Bradley, Burak Yavuz, Evan R. Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, D. B. Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Zaharia, and Ameet Talwalkar. 2015. MLlib: Machine Learning in Apache Spark. *CoRR* (2015).

- [139] Zachary Miller, Brian Dickinson, William Deitrick, Wei Hu, and Alex Hai Wang. 2014. Twitter Spammer Detection Using Data Stream Clustering. *Inf. Sci.* 260 (March 2014), 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*, Juliana Freire, David Koop, and Luc Moreau (Eds.). Lecture Notes in Computer Science, Vol. 5272. Springer Berlin Heidelberg, 253–265.
- [141] Rene Mueller, Jens Teubner, and Gustavo Alonso. 2009. Streams on Wires: A Query Compiler for FPGAs. *Proc. VLDB Endow.* 2, 1 (Aug. 2009), 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)*. Springer-Verlag, London, UK, UK, 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 Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13)*. ACM, New York, NY, USA, 439–455.
- [144] Christopher Mutschler and Michael Philippsen. 2014. Adaptive Speculative Processing of Out-of-Order Event Streams. *ACM Trans. Internet Technol.* 14, 1, Article 4 (Aug. 2014), 24 pages.
- [145] Zubair Nabi, Eric Bouillet, Andrew Bainbridge, and Chris Thomas. 2014. Of Streams and Storms. *IBM White Paper* (2014).
- [146] Roshan Naik and Sapin Amin. 2016. Microbenchmarking Apache Storm 1.0 Performance. URL: <http://hortonworks.com/blog/microbenchmarking-storm-1-0-performance/>, (2016).
- [147] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari. 2010. S4: Distributed Stream Computing Platform. In *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*. 170–177.
- [148] Odysseas Papapetrou, Minos Garofalakis, and Antonios Deligiannakis. 2012. Sketch-based Querying of Distributed Sliding-window Data Streams. *Proc. VLDB Endow.* 5, 10 (June 2012), 992–1003.
- [149] A. Pavan, Kanat Tangwongsan, Srikanta Tirhapura, and Kun-Lung Wu. 2013. Counting and Sampling Triangles from a Graph Stream. *Proc. VLDB Endow.* 6, 14 (Sept. 2013), 1870–1881.
- [150] Srinath Perera. 2013. CEP Performance: Processing 100k to Millions of events per second using WSO2 Complex Event Processing (CEP) Server. URL: <http://srinathview.blogspot.com/2013/08/cep-performance-processing-100k-to.html>, (2013).
- [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, USA, 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) - Workshopband, 2.-3. März 2015, Hamburg, Germany*. 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, USA, 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, USA, 101–112.
- [155] Chris Raphael. 2014. IDC Reveals Worldwide Internet of Things Predictions for 2015. URL: <https://www.idc.com/getdoc.jsp?containerId=prUS25291514>. (December 2014).
- [156] Chris Raphael. 2015. IoT Architectures for Edge Analytics. URL: <http://www.rtinsights.com/iot-architectures-for-edge-analytics/>. (December 2015).
- [157] Sajith Ravindra and Miyuru Dayarathna. 2015. Distributed Scaling of WSO2 Complex Event Processor. URL: <http://wso2.com/library/articles/2015/12/article-distributed-scaling-of-wso2-complex-event-processor/>. (December 2015).
- [158] Vlad Rozov. 2015a. Apache Apex Performance Benchmarks. URL: <https://www.datatorrent.com/blog/blog-apex-performance-benchmark/>, (2015).
- [159] Vlad Rozov. 2015b. Apache Apex Performance Benchmarks. URL: <https://www.datatorrent.com/blog/blog-apex-performance-benchmark/>. (October 2015).
- [160] RuleCore. 2015. Complex Event Pattern Detector. URL: <http://sourceforge.net/projects/rulecore/>. (August 2015).
- [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, USA, 187–198.

- [162] Bobby Evans Reza Farivar Tom Graves Mark Holderbaugh Zhuo Liu Kyle Nusbaum Kishorkumar Patil-Boyang Jerry Peng Sanket Chintapalli, Derek Dagit and Paul Poulosky. 2015. Benchmarking Streaming Computation Engines at Yahoo! URL: <https://yahoeng.tumblr.com/post/135321837876/benchmarking-streaming-computation-engines-at>. (December 2015).
- [163] Ivo Santos, Marcel Tilly, Badrish Chandramouli, and Jonathan Goldstein. 2013. DiAl: Distributed Streaming Analytics Anywhere, Anytime. *Proc. VLDB Endow.* 6, 12 (Aug. 2013), 1386–1389.
- [164] Ahmet Erdem Sarıfıyıcı, Buğra Gedik, Gabriela Jacques-Silva, Kun-Lung Wu, and Ümit V. Çatalyürek. 2013. Streaming Algorithms for K-core Decomposition. *Proc. VLDB Endow.* 6, 6 (April 2013), 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, USA, 53–64.
- [166] W. R. Schulte. 2015. CEP Technology: EPPs, DSCPs and other Product Categories. URL: www.complexevents.com/2015/07/10/cep-technology-epps-dscps-and-other-product-categories. (July 2015).
- [167] Nicholas Poul Schultz-Møller, Matteo Migliavacca, and Peter Pietzuch. 2009. Distributed Complex Event Processing with Query Rewriting. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems (DEBS '09)*. ACM, New York, NY, USA, 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, USA, 301–312.
- [169] Satoshi Oda Shohei Hido, Seiya Tokui. 2013. Jubatus: An Open Source Platform for Distributed Online Machine Learning. 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 Comput. Surv.* 46, 1, Article 13 (July 2013), 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, USA, 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 (3 2015), 213–245. Issue Special Issue 02.
- [173] Chunyao Song, Tingjian Ge, Cindy Chen, and Jie Wang. 2014. Event Pattern Matching over Graph Streams. *Proc. VLDB Endow.* 8, 4 (Dec. 2014), 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, USA, 20–31.
- [175] Robert Soul, Martin Hirzel, Bura Gedik, and Robert Grimm. 2015. River: an intermediate language for stream processing. *Software: Practice and Experience* (2015), n/a–n/a.
- [176] SQLstream. 2014. SQLstream Blaze and Apache Storm: A BENCHMARK COMPARISON. (2014).
- [177] SQLstream. 2015. SQLstream - Big Data Stream Processing & Operational Intelligence for the Internet of Things. URL: <http://www.sqlstream.com/>. (August 2015).
- [178] SQLstream. 2016. SQLstream Blaze Outperforms Apache Storm in Stream Processing Benchmark. URL: <http://sqlstream.com/2014/11/sqlstream-blaze-over-100x-faster-than-apache-storm-in-industry-benchmark-for-stream-processing-performance/>. (October 2016).
- [179] Utkarsh Srivastava and Jennifer Widom. 2004. Flexible Time Management in Data Stream Systems. In *Proceedings of the Twenty-third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '04)*. ACM, New York, NY, USA, 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, USA, 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, USA, 43–50.
- [182] Dawei Sun, Guangyan Zhang, Songlin Yang, Weimin Zheng, Samee U. Khan, and Keqin Li. 2015a. Re-Stream: Real-time and energy-efficient resource scheduling in big data stream computing environments. *Information Sciences* 0 (2015), –.
- [183] Dawei Sun, Guangyan Zhang, Weimin Zheng, and Keqin Li. 2015b. Key Technologies for Big Data Stream Computing. *Big Data: Algorithms, Analytics, and Applications* (2015), 193.

- [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 (2014), 181–197.
- [185] Richard Tibbetts. 2009. *Performance & Scalability Characterization*. StreamBase.
- [186] TIBCO. 2014. TIBCO StreamBase versus Native Threading. URL: <http://www.tibco.com/assets/blt270a958cb7af56af/wp-tibco-streambase-versus-native-threading-tcm8-20212.pdf>. (2014).
- [187] TIBCO. 2016. StreamBase Studio. URL: <http://www.streambase.com/products/streambasecep/streambase-studio/>. (January 2016).
- [188] Srikanta Tirthapura, Bojian Xu, and Costas Busch. 2006. Sketching Asynchronous Streams over a Sliding Window. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Principles of Distributed Computing (PODC '06)*. ACM, New York, NY, USA, 82–91.
- [189] Ankit Toshniwal, Siddharth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M. Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, Nikunj Bhagat, Sailesh Mittal, and Dmitriy Ryaboy. 2014. Storm@Twitter. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD '14)*. ACM, New York, NY, USA, 147–156.
- [190] Thanh T. Tran, Liping Peng, Yanlei Diao, Andrew Mcgregor, and Anna Liu. 2012. CLARO: Modeling and Processing Uncertain Data Streams. *The VLDB Journal* 21, 5 (Oct. 2012), 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, USA, 23–34.
- [192] Kostas Tzoumas, Stephan Ewen, and Robert Metzger. 2015. High-throughput, low-latency, and exactly-once stream processing with Apache Flink. URL: <http://data-artisans.com/high-throughput-low-latency-and-exactly-once-stream-processing-with-apache-flink/>. (August 2015).
- [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, USA, 120–129.
- [194] Paul Vincent. 2016. CEP Tooling Market Survey 2016. URL: <http://www.complexevents.com/2016/05/12/cep-tooling-market-survey-2016/>. (May 2016).
- [195] Y. Wang and K. Cao. 2012. Context-aware Complex Event Processing for event cloud in Internet of Things. In *Wireless Communications Signal Processing (WCSP), 2012 International Conference on*. 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 & Mathematics with Applications* 66, 10 (2013), 1808 – 1821. ICNC-FSKD 2012.
- [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, USA, 307–318.
- [198] Jielong Xu, Zhenhua Chen, Jian Tang, and Sen Su. 2014. T-Storm: Traffic-Aware Online Scheduling in Storm. In *Distributed Computing Systems (ICDCS), 2014 IEEE 34th International Conference on*. 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 *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*. 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 Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13)*. ACM, New York, NY, USA, 423–438.
- [201] Erik Zeitler and Tore Risch. 2011. Massive scale-out of expensive continuous queries. *VLDB Endowment* 4, 11 (2011).
- [202] Chunwang Zhang and Ee Chien Chang. 2014. Processing of Mixed-Sensitivity Video Surveillance Streams on Hybrid Clouds. In *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*. 9–16.
- [203] Haopeng Zhang, Yanlei Diao, and Neil Immerman. 2010. Recognizing Patterns in Streams with Imprecise Timestamps. *Proc. VLDB Endow.* 3, 1-2 (Sept. 2010), 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, USA, 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 *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*. 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 ISWC 2012. Lecture Notes in Computer Science*, Vol. 7650. Springer Berlin Heidelberg, 257–273.