Event Stream Processing: A Literature Review

Chih-Chieh Lin

University of Amsterdam
Vrije Universiteit Amsterdam
c2.lin@student.vu.nl

Abstract

As real-time and Internet of Things applications thrive in decades, the demand and standard for event stream processing (ESP) have been increased correspondingly in order to support such applications. Although ESP systems have been evolved for decades since 2000s, there still are limitations and challenges. This paper first defines the concept of ESP and evaluates the key challenges in ESP systems, along with two topics regarding modern ESP frameworks and future direction of ESP. This work discusses each topic by reviewing relevant academic literature and relevant online document for ESP frameworks, followed by conclusion summarizing the answers for each research question.

1 Introduction

For large number of applications in modern technology era, real-time processing or real-time response are essential since such applications cannot afford long latency or delayed processing time. For example, detecting plummeting or sudden increase in stock markets, anomaly indicator in patients' health care system, real-time traffic condition report, require real-time data processing and response in order to meet the need from users or customers. However, the real-time data is not stable or even stored in a static database. On the contrary, the real-time data usually is streaming data which is generated by data sources and transported along the networks. Also, as Internet of Things (IoT) applications thrive these years, data generated from sensors source are regarded as complex patterns [7], so that it need complex event processing [24] which is one kind of ESP. Therefore, proper and powerful ESP frameworks play decisive roles in supporting these applications.

Although IoT and real-time applications needed the ESP to support their functions, developing an ESP system is hard and should consider plenty of perspectives. As Stonebraker et al. [20] mentioned in their paper, 8 requirements are asked to satisfied in order to develop a real-time stream processing system. Besides the basic requirements for ESP systems, Fragkoulis et al. [12] compared the early and modern ESP systems in different perspectives (regarding time management, elasticity, fault-tolerance). Also, they pointed out the open problems in such perspectives which indicates that some of the challenges and the difficulty in developing ESP still can be improved.

Plenty of ESP frameworks or solutions had been developed by previous researchers. These ESP frameworks had evolve from early ones(e.g. Aurora, NigaraCQ , STREAM [6, 11, 9] etc.) to modern ones (Apache Storm, Apache Flink, Spark Streaming [4, 1, 5]). We are now still witnessing the emerging of ESP systems since the researchers are still improving ESP to overcome such challenges and limitations.

This paper is structured in four sections. First, the ***Introduction*** is provided to introduce the background of event stream processing. A ***Methodology*** followed by Introduction states the research questions of this literature review and corresponding motivation of each research question. Then in ***Discussion of Literature***, the contents are divided into three part in order to answer each research question respectively. Last but not least, a ***Conclusion*** is stated to summarize this paper.

2 Methodology

In order to clarify the direction of the literature review, research questions should be stated. Therefore, the research questions and the rational motivation for research questions are as follows.

- RQ1: *What are the future directions of development of the event stream processing framework?*
  - I desire to figure out what the future direction of ESP is. To be more specific, I try to summarize the future direction for researcher and developer based on current frameworks of ESP. To answer this research question, I must first obtain the fundamental knowledge and comprehensive background in ESP. Therefore, I further come up with two sub-questions as follows.
- RQ1.1: *What is event stream processing? What are the key challenges of event stream processing?*
  - I try to investigate the definition of event stream processing from a view of reader without any technical background. After setting up an initial definition, as a beginner of ESP, I try to identify the key challenges in this field.
- RQ1.2: *What are the modern frameworks for event stream processing?*
  - After having fundamental knowledge about ESP, I try to investigate how ESP system work. Therefore, I single out several representative current frameworks used for event stream processing, in order to figure out how they work and what their strategy and components are.

3 Discussion of Literature

3.1 Definition

Before discussing other perspectives of event stream processing, we must first understand what event stream processing is. An easy way to describe event

stream processing is that the processing or the analyzing of continuous streams of events. If we separate the terms and explain each term respectively, the "event" represents the data point in a system (e.g. patients' anomaly condition detection in health care system, plummeting detection in the stocks market). "stream" represents continuous data delivery of these events. "processing" refers to any action taken on these events or data (e.g. analyzing, transformation, aggregation).

Moreover, it is important that we must recognize the difference between data (batch) processing and event stream processing. Abadi et al.[6] describe their data stream management model, Aurora, as a "software system processing and reacting to continual inputs from many sources (e.g., sensors) rather than from human operators." However, traditional database management system (DBMS) is a passive system storing large static data elements lets human to take actions. Therefore, we can refer event stream processing as taking action on continual data flow while refer batch processing as taking action on static data.

Type of Stream Processing System The term event stream processing is not a simple method or a concept. When mentioning stream processing, we discuss the systematic approach way to solve the stream processing problem. Therefore, we provide the definition of the type of stream processing system. To be more specific, the type of stream processing system determines what operations or functions that the system provides. Röger and Mayer [17] distinguish General Stream Processing systems (GP) and Complex Event Processing systems (CEP) [24]. We illustrate the characteristics of two type of system in the Table 1.

3.2 Key Challenges in ESP

As ESP is a dynamic framework applying continuous operations from the input stream data instead of a static system allowing user to query and access. Therefore, the researchers may confront more complex and intractable problems when building or maintaining a ESP framework. Macrometa[1], a company providing platform for building real-time distributed applications, pointed out challenges of ESP in different perspectives. I picked several aspects from their contents. Moreover, some perspectives proposed by other researchers are also relevant and must be considered. Therefore, I summarize their viewpoints for key challenges of ESP in 4 aspects: QoS, time management, parallelization and elasticity. The rationale for selecting QoS is that QoS is the main performance evaluating mechanism for ESP systems. The other 3 aspects are discussed in the ESP field by researchers. Also, several comprehensive surveys [10, 12, 17] included these three aspects in their works. These are the rationale for selecting these four aspects. Moreover, more details for these aspects are provided as follows.

[1] https://www.macrometa.com/

Table 1: Type of stream processing system

Type	**Characteristic**	**Examples**
GP	- GP systems **continuously take actions** on the items of streaming data. - The output data streams are produced by the operators in the system. Each output stream can be an input stream for another operator, or can be used by other applications directly. - Data Stream Management System (DSMS) is provided by GP systems in order to **make ongoing queries** on streaming data.	Aurora, Apache Storm
CEP	- CEP systems are used for **detecting specific patterns** of data stream so that certain information is able to obtain. - The inputs of systems are composed of certain observed **triggered events**. - The operators would take charge of finding the series of ongoing data stream which satisfies the certain pattern. - Whenever a pattern is detected, the operator emits an output event.	Apache Flink

- Quality of Service (QoS)

It is always important to pursue high QoS or to provide users with high QoS in ESP systems. QoS is a performance measure for evaluating ESP systems' performance. QoS can be separated into two parts - **throughput** and **latency**. High throughput and low latency are the main QoS goals for ESP systems. Shulka et al. [18] define latency in ESP systems as the interval between inputs consumed at the source tasks and all its output messages generated at the sink tasks. Also, they define throughput as the aggregated rate of output messages emitted from the output operators (sink tasks) in a certain interval.

It is hard to maintaining high QoS in ESP systems since it not only requires well-equipped infrastructure but also strategies for resource management (elasticity) and parallelization. Moreover, there may be a trade off between low latency and high throughput [23]. All of the perspective mentioned above are thorny problems which make pursuing high QoS harder.

- Time management

On one hand, in ESP systems, the timestamp or the notion of time must be defined clearly. Otherwise, it will affect the quality of output results directly. On the other hand, the stream data can be out-of-order since they may arrive ESP system with different latency (late data) [19]. Stonebraker et al. [20] illustrates the same concept with an example - the data streams reaching in certain time window are marked by a corresponded timestamp, while some data arrives lately (out-of-order). Therefore, handling out-of-order and manage the correct timestamp are crucial issue in ESP systems.

- Parallelization

As for parallelization, Röger and Mayer [17] distinguish two cases of parallelization: **task parallelization** and **data parallelization**. In task parallelization, ESP systems parallelize the different operators, while in data parallelizations, they partition the input stream with different methods and then run multiple instances of identical operator.

Röger and Mayer [17] also proposed the challenges and limitations of each case of parallelization. The main challenge on task parallelization is that it will reveal three limitations. First, the network traffic increases when multiple operators receive the input streams completely. Second, if the operators processing speeds differ, the load balance may be affected. Third, the scalability in task parallelization is limited.

For the challenge of data parallelization, they discuss the limitations with different methods of data splitting. To be short, the limitations of data parallelization include expressiveness, scalability, load balancing, and increasing communication overhead, which depend on different splitting method.

- Elasticity

Elasticity or resource elasticity is a key characteristic in cloud computing, which allows organisations to dynamically change infrastructure capacity with minimum human intervention, i.e. auto-scaling systems. Lorido-Botran et al. [16] pointed that while it enables a service to allocate and release resources dynamically, adjust the changing demands, determining the proper resources is hard. de Assunção et al. [10] further classified the managing elasticity of data stream processing into two problems: (1) releasing and allocating the resources in order to coordinate with the workload; and (2) arranging and performing the actions for adjusting the workload to get use of newly allocated or released resources.

3.3 Modern ESP Systems

Generation of Streaming Processing Systems Before illustrating the content of modern ESP systems, it is better to realize the generations of SP systems, so that we can have a comprehensive view of emerging and developing trend of SP systems. de Assunção et al. [10] classified the emerging of SP engines into four generations. Moreover, Fragkoulis et al. [12] provides a comprehensive view of evolution of stream processing systems. I summarize their point of view and provide them in Table 2.

Current ESP Systems The third generation's stream processing systems is discussed in the following section, since we desire to focus on the state of the art solution of stream processing. However, the first and second generation's SP systems are obsolete because most of them are developed over a decade ago. Moreover, the fourth generation's SP systems are still in developing (emerging) period which are not mature enough. Therefore, we focus on third generation which is well-developed, and single out several representative framework to discuss.

- Apache Storm

A data pipeline in Apache storm [4] is also called **Topology**. Fig 1(a) shows the concept of topology, which represents a directed graph of data flow and depicts how the tuples (data) process among them. Apache storm utilize master-slave structure with a master node running daemon process called **Nimbus**. Nimbus interacts with **Zookeeper** [13] to schedule the tasks among clusters and take actions for node failures. Although Apache Storm is fault-tolerant and scalable, Wingerath et al. [23] pointed out that its state management can only feasible on application with small state, due to synchronous updating which will affect latency. Moreover, Apache Storm implements a back pressure mechanism to throttle data ingestion in case data ingested speed overtakes the processing speed.

A storm cluster is composed of several worker nodes. Each worker node runs a supervisor daemon process and several worker processes (i.e. JVM [2] [22]). A

[2] Java Virtual Machine

Table 2: Generations of stream processing system

Generation	**Characteristic**
First	- The first generations's SP system supports extension of classical DBMS. - Allow queries over dynamic data. - Provide interface and SQL-like language which allow users for operations.
Second	- The second generation's SP system enable distributed processing by using message passing for communicating. - The distributed processing, however, reveals the challenge for elasticity management and load balancing.
Third	- Enable specification and execution of user defined functions (UDF). - Ordinarily, deploy on shared-nothing clusters. - Have task and data parallelization. - Enable explicit state management and enhanced fault tolerance.
Fourth	- Some processing elements are placed at the edge of the network. - More distributed environment (fog, edge computing). - Programmable network - In-transit processing systems - Online event processing [14] - Heterogeneous architectures [15]

JVM runs several executors which execute tasks (spouts and bolts). A **spout** takes charge of reading the streaming data from an external source. A **bolt** is responsible for listening to data from internal nodes and writing data to the external storage.

- Apache Samza

Apache Samza [3] is a data stream framework which is co-developed with Apache YARN [21] for deployment and with Apache Kafka [2] for messaging. A Samza job submitted through YARN client triggers YARN to initiate and supervise one or more containers. Apache Samza jobs are not able to communicate directly, but can use Kafka as message broker. An example of a samza job illustrates at Fig 1(b), which represents a step in processing pipeline and is corresponded to the bolt in Apache storm.

In comparison to Apache storm, Wingerath et al. [23] pointed out that Apache Samza needs more work to deploy since it does not only depend on Zookeeper but also requiring Apache YARN for fault tolerance. Apache Samza also support JVM-Language, and are also scalable since a samza job runs in parallel tasks by Kafka partitions. For state management, contrasting to Apache Storm, Apache Samza can handle large amount of state in fault tolerance since it maintains the local state and replicates them to Kafka. A key-value store is utilized for recording this state updates in other machines in cluster, so that it can recover quickly from failure.

- Apache Flink

Apache Flink [1] supports batch and data stream processing in common runtime. Apache Flink execution model is also a master-slave execution architecture, which a master called **Job Manager** and workers called **Task Managers**. The job manager takes charge of checkpointings (watermark items are injected into data stream in order to create states' checkpoints), failure recovery, receiving jobs, scheduling tasks. The task managers executes the tasks in the data flow of Flink. In Flink applications, Operators can be divided into subtasks (which are allocated in different containers or machines) and streams can also be separated into stream partitions. Therefore, Flink can achieve parallelization through corresponded two mechanisms: determining degree of parallelism of operators and streams.

Analogous to storm, the workers in Flink support JVM process which allows them run subtasks in separated thread. Also, similar to Storm, Flink implement a back pressure mechanism through buffering in order to avoid ingestion overtaking the processing speed.

- Spark streaming

Apache Spark is a cloud computing framework, which is an extension of MapReduce model, supports streaming and batch processing. In a Spark deployment, a cluster manager is responsible for managing resources; a driver program takes charge of scheduling; several workers run the tasks for execution.

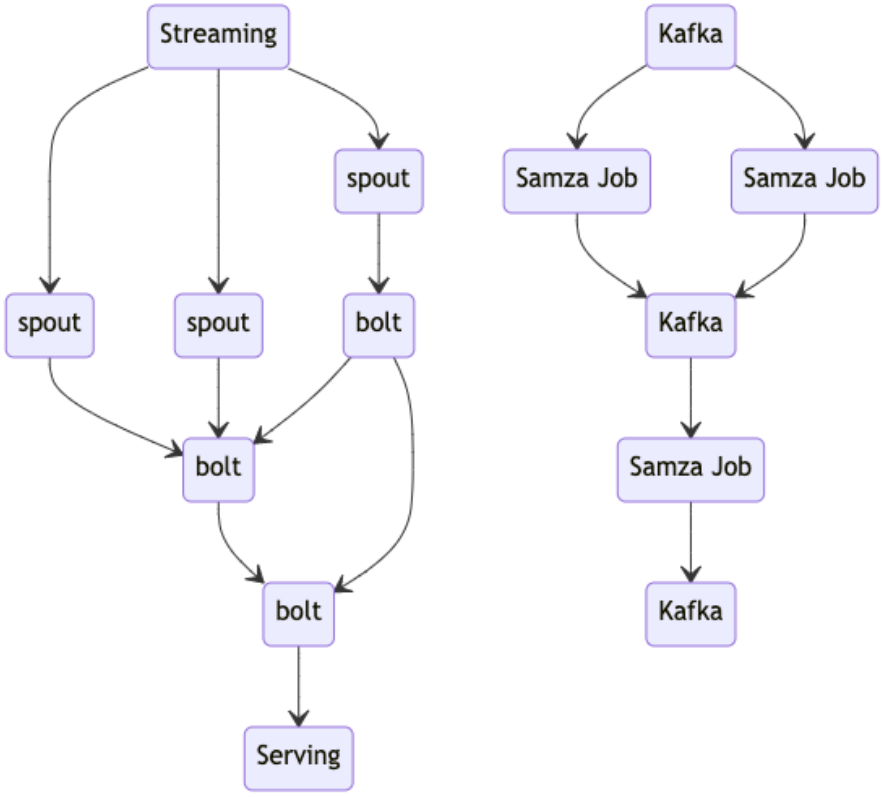

Fig. 1: (a) Apache Storm topology and (b) Apache Samza data flow [23]

Apache Spark introduces a core abstraction called **RDD** (resilient distributed datasets), which are distributed and immutable collections of records. Utilizing RDD enables Spark to keep track of RDD's lineage so that it is resilient to fault tolerance or machine failure.

Moreover, in Spark, before processing through workers, data is transformed into sequence of RDD called **DStream** [25] (discretized stream) in order to better deal with faults and stragglers. In a short interval, Dstream store the receiving data and generate new datasets representing intermediate state. The intermediate state is composed of RDDs and datasets stored during the next interval. This model stores the states in the memory so that it can recompute and resume fast from faults.

3.4 The Future Direction in ESP

As de Assunção et al. [10] stated in their survey that we are now witnessing the emergence of fourth generation of data stream processing systems, highly distributed frameworks using fog and edge computing are explored currently. They also proposed several directions which may encourage researchers and developers to focus on. First, using Software Defined Network (SDN) and Network Functions Virtualization (NFV) not only makes network programmable but also provides approach to allocate network capacity for data flow among data centres. Second, *in-transit* stream processing which places the processing elements or operators along the network between data source and the cloud. Third, as launching of high-level abstraction frameworks (e.g. Google cloud dataflow [8]) and large amount of demands in IoT application, researchers have tried to find solution to reduce latency in the field of edge and fog computing.

Moreover, the key challenges mentioned in section 3.2 must be considered. It is always beneficial to pursue high QoS. However, there may be a trade off between low latency and high throughput [23]. For time management, managing out-order data stream is still a thorny problem nowadays. (e.g. aligning watermarks in different time domains from various data sources) As for parallelization and elasticity, Röger and Mayer [17] also pointed out the trend of edge, fog, in-network computing. They encouraged considering such computing with heterogeneous environment.

4 Conclusion

As the evolution of event stream processing in decades, the researchers and developers improve the ESP systems from one generation to the next. The challenges and limitations for devising ESP systems are considerable, so only several representative key challenges are included in the work. Also, a large amount of ESP frameworks had been launched with strategy confront these challenges while I single out four frameworks to introduce their characteristics and details. To conclude this work, I try to summarize the answers for the main research question and the sub-questions as follows.

- RQ1: *What is the future direction of development of the event stream processing framework?*

It is hard to identify a specific future direction for ESP since the current frameworks still confront the challenges and limitations. Some researchers provided their views and encourage exploring the edge computing for the next generation of ESP. As a result, with edge computing, the frameworks will be more distributed by placing multiple data centres and processing elements in them.

Also, software define framework and network functions virtualization producing programmable networks benefit the data flow between data centres. Programmable networks provide the ability to allocate network capacity among data centres.

In-transit stream processing is another issue or direction to investigate. It places the processing element along the network between source and the cloud.

Moreover, some researchers focus on the online event processing [14] in order to solve the problems in transaction-processing domain.

In addition, the edge computing combined with heterogeneous environment can be another direction to explore. The directions mentioned above are relevant to the trend in forth generation framework provided in Table 2. Finally, I list them as follows in order to provide a concise overview of these directions.

- Edge computing for more distributed environment
- Programmable networks
- In-transit processing systems
- Online event processing
- Heterogeneous architectures [15]

- RQ1.1: *What is event stream processing? What are the key challenges of event stream processing?*

A clear definition of event stream processing is provided followed by a classification (type) of ESP systems. This way, we can not only understand what ESP is, but also recognize what ESP systems do and what their difference are. The characteristics and example frameworks of two types of ESP systems (GP and CEP) are provided. Moreover, 4 key challenges for ESP are selected: quality of service, storage, time management, parallelization, and elasticity. They are all relevant and representative not only for previous ESP systems but also for current ESP frameworks. For QoS, a performance mechanism for ESP systems, there may be a trade off between high throughput and low latency. For storage, the ESP should support various operations to maintain flexibility and efficiency. For time management, it is hard to deal with out-of-order data. For parallelization, each case of parallelization confront different limitations, such as traffic increasing in network, scalability and load balancing etc. For elasticity, it is hard to plan a allocate and release strategy; also hard to do such actions for the strategy.

- RQ1.2: *What are the modern frameworks for event stream processing?*

The Table 2 introduces the generations of ESP system which give us an overview of evolution of ESP frameworks. To be short, the first generation is the extension of DBMS; the second generation start focusing on distributed work; the third one enables the user defined functions and supports explicit state management; the forth are still emerging and focusing on the edge computing field.

Four current ESP frameworks are illustrated in section 3.3, which are Apache Storm, Apache Samza, Apache Flink, Spark Stream. Each of them are representative for modern ESP frameworks. For Storm, a topology concept and the executors: spouts and bolt, should be emphasized. For Samza, the deployment system: YARN, and the message broker: Kafka, are its important features. For Flink, a job manager, task managers, and checkpointing with watermarks are impressive. For spark streaming, it introduce RDD and DStream which make ESP more reliable and fault-tolerant.

References

1. Apache Flink. `https://flink.apache.org/`
2. Apache Kafka. `https://kafka.apache.org/`
3. Apache Samza. `https://samza.apache.org/`
4. Apache Storm. `https://storm.apache.org/`
5. Spark Streaming. `https://spark.apache.org/docs/2.0.0/streaming-programming-guide.html`
6. Abadi, D.J., Carney, D., Cetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N., Zdonik, S.: Aurora: a new model and architecture for data stream management. the VLDB Journal **12**(2), 120–139 (2003)
7. Akbar, A., Khan, A., Carrez, F., Moessner, K.: Predictive analytics for complex iot data streams. IEEE Internet of Things Journal **4**(5), 1571–1582 (2017)
8. Akidau, T., Bradshaw, R., Chambers, C., Chernyak, S., Fernández-Moctezuma, R.J., Lax, R., McVeety, S., Mills, D., Perry, F., Schmidt, E., et al.: The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing (2015)
9. Arasu, A., Babcock, B., Babu, S., Datar, M., Ito, K., Nishizawa, I., Rosenstein, J., Widom, J.: Stream: the stanford stream data manager (demonstration description). In: Proceedings of the 2003 ACM SIGMOD international conference on Management of data. pp. 665–665 (2003)
10. de Assuncao, M.D., da Silva Veith, A., Buyya, R.: Distributed data stream processing and edge computing: A survey on resource elasticity and future directions. Journal of Network and Computer Applications **103**, 1–17 (2018)
11. Chen, J., DeWitt, D.J., Tian, F., Wang, Y.: Niagaracq: A scalable continuous query system for internet databases. In: Proceedings of the 2000 ACM SIGMOD international conference on Management of data. pp. 379–390 (2000)
12. Fragkoulis, M., Carbone, P., Kalavri, V., Katsifodimos, A.: A survey on the evolution of stream processing systems. arXiv preprint arXiv:2008.00842 (2020)
13. Hunt, P., Konar, M., Junqueira, F.P., Reed, B.: Zookeeper: Wait-free coordination for internet-scale systems. In: USENIX annual technical conference. vol. 8 (2010)
14. Kleppmann, M., Beresford, A.R., Svingen, B.: Online event processing. Communications of the ACM **62**(5), 43–49 (2019)

15. Koliousis, A., Weidlich, M., Castro Fernandez, R., Wolf, A.L., Costa, P., Pietzuch, P.: Saber: Window-based hybrid stream processing for heterogeneous architectures. In: Proceedings of the 2016 International Conference on Management of Data. pp. 555–569 (2016)
16. Lorido-Botran, T., Miguel-Alonso, J., Lozano, J.A.: A review of auto-scaling techniques for elastic applications in cloud environments. Journal of grid computing **12**(4), 559–592 (2014)
17. Röger, H., Mayer, R.: A comprehensive survey on parallelization and elasticity in stream processing. ACM Computing Surveys (CSUR) **52**(2), 1–37 (2019)
18. Shukla, A., Simmhan, Y.: Benchmarking distributed stream processing platforms for iot applications. In: Technology Conference on Performance Evaluation and Benchmarking. pp. 90–106. Springer (2016)
19. Srivastava, U., Widom, J.: Flexible time management in data stream systems. In: Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. pp. 263–274 (2004)
20. Stonebraker, M., Çetintemel, U., Zdonik, S.: The 8 requirements of real-time stream processing. ACM Sigmod Record **34**(4), 42–47 (2005)
21. Vavilapalli, V.K., Murthy, A.C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., et al.: Apache hadoop yarn: Yet another resource negotiator. In: Proceedings of the 4th annual Symposium on Cloud Computing. pp. 1–16 (2013)
22. Venners, B.: The java virtual machine. Java and the Java virtual machine: definition, verification, validation (1998)
23. Wingerath, W., Gessert, F., Friedrich, S., Ritter, N.: Real-time stream processing for big data. it-Information Technology **58**(4), 186–194 (2016)
24. Wu, E., Diao, Y., Rizvi, S.: High-performance complex event processing over streams. In: Proceedings of the 2006 ACM SIGMOD international conference on Management of data. pp. 407–418 (2006)
25. Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., Stoica, I.: Discretized streams: Fault-tolerant streaming computation at scale. In: Proceedings of the twenty-fourth ACM symposium on operating systems principles. pp. 423–438 (2013)