Complex Event Processing for the Internet of Things

Ariane Ziehn
Supervised by Volker Markl and Steffen Zeuch
Technical University of Berlin,
German Research Center for Artificial Intelligence
ariane.ziehn@dfki.de

ABSTRACT

Complex Event Processing (CEP) enables autonomous and real-time decision making in data management systems. Today, applications leverage CEP only in cloud-based environments to provide prompt reactions, although data are generated outside the cloud. In particular, the Internet of Things (IoT) will increase the number of data producers that a single IoT application has to handle millions of devices. The centralized data collection before applying data processing introduces a critical bottleneck in current cloud-based solutions, especially for delay-sensitive IoT applications. To overcome this bottleneck, fog computing emerged as a paradigm to process data close to the network edge. However, CEP systems are not yet ready to leverage the fog layer as an extension for cloud-based stream processing. In this paper, we examine how the current system has to adapt to exploit the new capabilities of the fog for CEP. To this end, we analyze principal CEP methodologies and propose new solutions. With this work, we lay the foundation for large-scale in-network CEP applications on top of the IoT.

1. INTRODUCTION

Complex Event Processing (CEP) is a common method for real-time stream processing to detect sequences of events in data streams and triggers actions upon detection [7, 14, 16]. User-defined rules specify both the events and the actions that enable autonomous real-time decision making in a wide range of applications, e.g., traffic congestion monitoring, live maps, intelligent transportation systems, smart street lamps, or vehicle pollution control [1, 8, 24]. Several cloud-based stream processing engines (SPEs) [4, 10, 21] provide CEP for rule-based monitoring as the current solution for the Internet of Things (IoT) scenarios with low-latency real-time response requirements. Under consideration of the constant increase of IoT devices, future IoT applications will process data from potentially millions of devices. Thus, cloud-based solutions are not capable of fulfilling the low-latency real-time response requirements due to the massive amount of data generated in these future IoT applications [12, 22]. Zeuch et al. [22] address this problem and propose a fog-cloud environment that leverages a fog layer as extension of the cloud. Figure 1 presents the data flow in both (a) cloud and (b) fog-cloud environment. On the left-hand side, the cloud environment collects data of connected

Proceedings of the VLDB 2020 PhD Workshop, August 31st, 2020. Tokyo, Japan. Copyright (C) 2020 for this paper by its authors. Copying permitted for private and academic purposes.

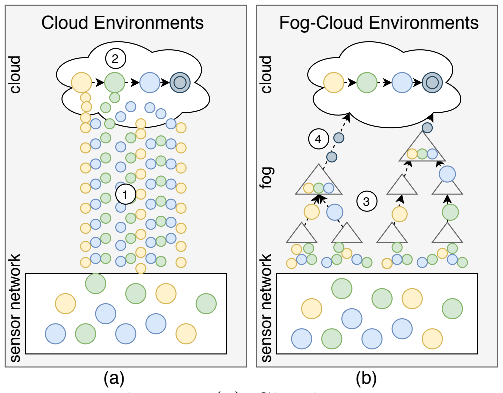


Figure 1: Data Flows in (a) Cloud Environments and (b) Fog-Cloud Environments.

devices centrally (1). Usually, the cloud environment uses a data broker like Kafka for buffering data. Then, the cloud-based SPE utilizes its almost unlimited resources to process the collected data in the cloud (2). However, the central data collection before processing is highly resource-intensive and causes significant delays as well as network overhead. Thus, cloud environments introduce a critical bottleneck for future IoT data management systems that must handle millions of data streams [15, 22]. On the right-hand side, the fog-cloud environment introduces an additional fog layer with a typical tree-like network topology (3). The fog nodes △ process and reduce data on the paths through the network (4). Hence, they mitigate the central bottleneck and release network capacities for low-latency real-time responses.

We argue that future IoT applications require a fog layer to process millions of data streams efficiently. Therefore, it is crucial to leverage fog environments for CEP and its capabilities to enable future IoT applications with millions of devices and thousands of rules. However, fog environments introduce new challenges for CEP, e.g., stateful in-network processing on low-end devices, changing network topologies, and mobile IoT devices. We investigate how to tackle these challenges in order to leverage the fog layer efficiently for CEP. In this paper, we make the following contributions to enable the IoT for CEP:

- We analyze state-of-the-art CEP systems and identify the major limitations to leverage the IoT as a mean for large-scale, in-network CEP.
- We highlight three concrete problems and sketch possible solutions to enable the IoT for CEP.

- We outline possible improvements to our approaches, which extend state-of-the-art CEP processing in the areas of pattern evaluation mechanisms, their optimization, and pattern specification languages.

In the remainder of this paper, we analyze state-of-the-art CEP concepts in Section 2. Then, we highlight what prevents CEP from leveraging the IoT and outline three concrete problems in Section 3. In Section 4, we summarize related work and conclude our findings in Section 5.

2. RESEARCH CONTEXT

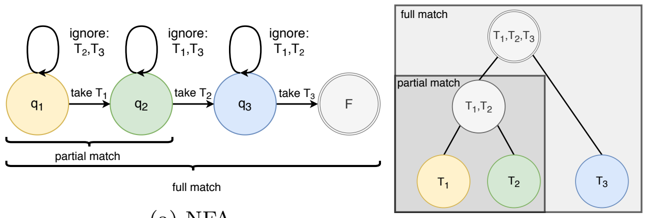
In this section, we introduce the concepts of rules and event patterns. Afterward, we show the state-of-the-art pattern evaluation and optimization mechanism for CEP.

Rules: Rules build the knowledge base of the CEP engine and are used in active database systems for autonomous reactions on data manipulations. For this purpose, the user specifies rules with up to three components, i.e., *event* e, *condition* c, and *action* a, which lead to the name ECA-rules. Each tuple manipulation represents an event e that might trigger the user-desired action a if e matches the relation-specific or inter-event conditions c. Actions are application-specific and can be user notifications such as warnings and alerts or autonomous system actions such as updating attribute values or triggering other rules [18].

Data Streams: We want to detect those rules in data streams, where each data stream E is a continuous and unbounded flow of data tuples. Each generated tuple is an event e that represents the state of its producer, e.g., an IoT device, at a specific point in time ts. Due to the increasing amount of geographically distributed IoT devices, the device location s is another essential tuple attribute. To sum up, each event source produces a stream E of events e_n with its set of attributes $E = \{ts, s, a_1, ..., a_n\}$, where ts and s are spatiotemporal attributes, and a_n are non-spatiotemporal attributes, e.g., measurements and identifiers [3].

Event Patterns: In SPEs, users define so-called simple event patterns [4] as a complement to traditional ECA-rules. A simple pattern is a $\langle e, c \rangle$ pair for one *event type* T. Event types are the replacement of relations in stream processing and provide a uniform schema $T = \{ts, s, a_1, ..., a_n\}$ for a group of contributing streams E_n [20, 24]. For instance, all sensors that measure the temperature at different locations contribute to the type T_{temp}. All arriving events are monitored with the event type-specific conditions c_{T_n}, and an action a is triggered if a matching event is detected.

Pattern Specification Language: The user formulates complex patterns of monitoring tasks as a composition of simple patterns from different event types. The relationships between simple patterns are defined by event operators, which are either logical, e.g., AND, OR, or temporal, e.g., the window operator $WITHIN$ or the sequence operator SEQ, which defines the order of simple patterns [20]. A wide range of pattern languages with different sets of event operators exists, e.g., SQL-like languages such as SASE+ [20, 23] and CCL [24], or languages based on event logic, e.g., CEL [6]. An example pattern for a vehicle pollution control application can be formulated as follows: *Detect if within 30 minutes an increased amount of vehicles (T_1) is followed by a decrease of the average speed (T_2), which leads to an excess of the air pollution level (T_3).* The example defines a sequence of three consecutive simple patterns for different event types T_n, including a condition c_{T_n} for each type. We use SASE+ [23] to specify the pattern in Listing 1.



(a) NFA.

(b) Tree.

Figure 2: Evaluations Models for Pattern Detection: NFA (a) and Tree Structure (b).

Listing 1: Vehicle Pollution Control Example

```
PATTERN  SEQ($T_1\,e_1, T_2\,e_2, T_3\,e_3$)
WHERE(   $e_1$.vehicleCount > $c_{T_1}$ AND
         $e_2$.speedLevel < $c_{T_2}$ AND
         $e_3$.pollutionLevel > $c_{T_3}$ )
WITHIN  TIME.MINUTES(30)
```

Pattern Evaluation Mechanisms: To detect matching events, SPEs create a pattern detection plan given the internal pattern representation. Common detection plans are tree-based plans [17], order-based plans with state machines [23], e.g., non-deterministic finite automaton (NFA), as well as event processing networks and graphs [6, 11]. We focus on the research lines with the highest number of representatives, tree-based and order-based evaluation plans [9]. Naive approaches of both lines represent the pattern identical to the user-given formulation and create one instance of the evaluation structure for each detected event. The detection of a complex pattern subsequence is denoted as a partial match, while the detection of a complete pattern is called a full match [13]. The example evaluation structures of an NFA in Figure 2a represents each partial match in one state q_n and a full pattern match in the final state F. The tree-based mechanism creates a leaf for each simple pattern (Figure 2b). Each intermediate node presents partial matches and the root a full match.

CEP belongs to the group of stateful query processing methods as SPEs need to store all partial matches either until the next partial match is detected or the window is expired. The number of partial matches is influenced by many factors, e.g., query selectivity or event frequency, but worst-case scenarios have exponential growth [14]. Therefore, pattern detection plan optimization aims to reduce the number of partial matches, e.g., by rewriting of single patterns or sharing techniques for multi-pattern CEP [14].

An effective rewriting method is called *Lazy NFA* [13]. It processes the pattern out-of-order by putting rare events in front of the pattern sequence. Frequent events are buffered and only analyzed after the rare event has been detected.

A new line of research focuses on the translation of patterns into multi-join queries. Thereby, patterns can be evaluated as stream queries and leverage existing join query optimizations [13]. For instance, we could rewrite the example pattern from Listing 1 (excluding $WITHIN$) by replacing the SEQ operator with the AND operator and adding additional inter-event constraints, as shown in Listing 2 [13, 24]. By replacing the temporal operator with a logical alternative, the pattern can be translated into a join query and profit from join-query optimizations.

Listing 2: CEP Traffic Pattern Example

```
PATTERN AND (T1 e1, T2 e2 , T3 e3)
WHERE( e1.vehicleCount > condition_T1 AND
       e2.speedLevel < c_T2 AND
       e3.pollutionLevel > condition_T3 AND
       e1.ts < e2.ts AND
       e2.ts < e3.ts)
```

3. LEVERAGE IOT FOR CEP

As opposed to fog-cloud environments, state-of-the-art SPEs process a global union of all sources E_n as one large stream. This processing strategy causes delays for modern IoT applications because it enforces the central data collection from millions of sensors before processing. Fog-cloud environments allow the processing of individual sensor streams E or subsets of event types T close to the data producers in the fog layer. Thus, this layer allows data reduction of more than 80% for stream queries, which reduces network traffic and enables the system to handle the data from millions of devices with low-latency [22].

Research Goal: *We aim to leverage fog environments for CEP and provide a solution that fulfills the low-latency and real-time response requirements of future IoT applications. To this end, we investigate the core features of CEP: pattern evaluation mechanisms, their optimization, and pattern specification languages.*

3.1 Pattern Evaluation Mechanisms

Efficient CEP requires a high-performance pattern evaluation mechanism. Currently, available evaluation mechanisms optimized for cloud-based environments could be applied to fog-cloud environments, yet without leveraging the fog layer, bottlenecks of cloud solutions would remain. Thus, the first problem we want to tackle in this work is:

Problem I: *Common cloud-based pattern evaluation mechanisms use a central component for data processing, pattern detection monitoring, or both. This central component prevents leveraging a fog environment without additional in-network distribution strategies.*

Opposed to the cloud paradigm, fog environments allow us to tailor the data to the relevant only on the paths through the network. As data is only shared with nodes on the network path, the pattern detection plan needs to be aware of the fog nodes that receive the relevant data to execute sub-plans. Further, by running sub-plans on fog nodes, we need to consider that CEP is a stateful processing method that needs to store partial matches on low-end devices.

Solution Sketch: We intend to identify promising evaluation mechanisms for distributed pattern detection and bring them together with the fog paradigm and distribution strategies. Since no general pattern evaluation mechanism with explicit performance guarantees exits, the selection of one research line for fog environments is not straight forward and requires an experimental evaluation. To this end, we consider all three approaches, order-based, tree-based, and pattern translation into multi-join queries (Sec. 2), as possible candidates. As the next step of our research agenda, we intend to implement a naive distributed solution for each of the three pattern evaluation mechanisms, including the necessary adaptions to leverage the fog layer. Afterward, we can compare our implementations using stream processing metrics, e.g., forward delays for matches. Additionally, the accuracy of our result in comparison with cloud solutions, where all data is centrally available, can be evaluated by accuracy metrics [9]. To this end, we can identify promising in-network evaluation mechanism for fog environments and optimize them further using our evaluation results in the following step of our research agenda.

3.2 Optimization of Evaluation Mechanisms

Storing and maintaining large amounts of partial matches is already a significant challenge in cloud environments with almost unlimited resources capacities. For fog environments, this challenge is even more critical as they contain low-end devices with limited capabilities but need to deal with partial matches of possibly hundreds of patterns. The heterogeneous hardware in fog environments lead to the second problem we want to tackle:

Problem II: *Cloud-based optimization techniques do not consider the limitations and challenges of unreliable and moving low-end devices and dynamic network topology.*

Both optimization techniques, rewriting of single patterns, and sharing techniques for multi patterns (Sec. 2), aim to improve the pattern detection plan in order to reduce partial matches. Cloud-based implementations of these strategies make assumptions that do not hold in fog environments. First, cloud-based SPEs examine the NP-complete problem of calculating one optimal pattern detection plan for their static network [7, 13]. In a dynamically changing network, the existing solutions would cause the expensive re-computation of new optimal plans after each topology change. Second, existing distributable solutions rarely consider heterogeneous hardware and resource limitations of low-end devices for distribution strategies [2, 19]. Third, to enable the rewriting technique out-of-order pattern detection, we need to store event buffers for retrospective pattern evaluation. Storing potentially high frequent events from hundred of producers challenges again the capacity limits of low-end devices and requires data compression.

Solution Sketch: With this work, we want to identify optimization techniques for distributed pattern evaluation on mobile low-end devices for hundreds of concurrently running patterns. First, we intend to prevent the NP-complete problem of finding one optimal solution and investigate strategies that identify sets of possible near-optimal pattern detection plans. These plans can be used as available fall-backs in case of topology changes. Second, in a fog-cloud environment, we can leverage the cloud as coordinator for pattern maintenance and distribution. By distributing stateful computation tasks of pattern evaluation to potentially mobile devices, an additional challenge appears: pattern evaluation might not be possible due to data producers. In this case, the user must be informed that currently, either monitoring is not possible or the results are probabilistic, e.g., derived from nearby sensors. Third, we want to investigate efficient compression techniques, e.g., partial aggregations [5], for event buffers on low-end devices to enable out-of-order pattern detection. Furthermore, the translation of patterns into multi-join queries enables another potential optimization strategy. In essence, we can combine both stream queries and pattern detection in one engine for optimization and maximize results sharing in fog-cloud environments. To this end, we leverage traditional CEP optimization strategies for low-end devices in a dynamic network and herewith enable efficient CEP for millions of devices and thousands of patterns.

3.3 Pattern Specification Language

Similar to evaluation mechanisms, no general pattern specification language exists, so no comprehensive set of event operators is available. Further, some languages lack formal semantics, provide limited expressiveness, or prevent automated optimization [6, 19, 20]. Besides, these languages are designed and optimized for single machines or cloud applications without considering future IoT applications. Thus, the third problem we want to tackle is:

Problem III: *Existing specification languages lack essential event operators because they were initially not intended for IoT applications. Further, many of them introduce restrictions that negatively impact efficient distributed CEP optimizations.*

We want to enable the formulation of complex patterns for IoT applications by identifying an easy-to-use specification language with the necessary set of event operators. For Problem I and II, we focus on the most common set of event operators, i.e., *AND*, *OR*, *NOT*, *SEQ* [6, 13] and extend it in this step of our research agenda.

Solution Sketch: Giatrakos et al. [11] reviewed the pattern specification languages of several CEP systems according to their expressiveness, including the Big Data SPE Apache Flink [4]. In contrast to other SPEs such as Apache Storm [10] or Spark [21], Flink provides built-in support for CEP [11] and additional operators compared to traditional single-machine CEP systems. However, Flink does not offer a specification language but provides an API with low-level functions. To this end, we use its pattern API as a baseline for our operator set and investigate how these operators can leverage a fog layer. Further, we build a pattern specification language on top of this operator set under the consideration of other leading languages, e.g., ZStream [17].

4. RELATED WORK

In this section, we summarize the state of the art of related work and highlight the major differences to our approach.

CEP Optimization: Kolchinsky and Schuster [13] proved that the pattern detection plan could be translated into multi-join queries and leverage join-query optimization. We utilize this result and consider multi-join queries as one possible evaluation mechanism for fog-cloud environments. Another novel approach proposed by Kolchinsky and Schuster [14] is the combination of rewriting and prefix sharing to optimize multi-pattern CEP. We intend to leverage a subset of these techniques for our approach.

In-network CEP: Madumal et al. [16] proposed a tree-based pattern evaluation approach for CEP with a rule engine that schedules the events either to a local CEP engine of a fog node or the Cloud CEP engine. As opposed to their approach, we focus on pattern evaluation in multiple nodes to leverage the tree-like topology of fog environments. Comet [7] is a decentralized ordered-based CEP approach for delay-tolerant networks. Akdere et al. [2] propose network-aware distribution strategies managed by a central control instance. Parts of both the approaches mentioned above can be reused for our CEP implementation. Nevertheless, in contrast to both, we focus on a solution that considers the limitations of low-end devices in a dynamic environment. Akili [3] motivated the need for decentralized CEP and proposed a tree-based approach for efficient multi-sink operator placement for non-hierarchical SPE. Multi-sink operator placement is a general stream processing problem and thus a complementing feature of our solution.

5. CONCLUSION

In this paper, we introduce and motivate our goal to enable efficient CEP for IoT data management systems in fog-cloud environments. We review the state-of-the-art solutions, identify their problems to leverage the fog layer, and suggest possible solutions. Further, we propose the following three steps to reach our goal: (I) identify and evaluate appropriate in-network pattern evaluation mechanisms for fog-cloud environments, (II) leverage and adapt optimization techniques for these mechanisms that fit the properties of low-end devices, and (III) build a pattern specification language for the IoT with CEP operators that leverage the fog layer. Next, we focus on Problem (I) and the implementation of evaluation mechanisms, including adaptions, to leverage the fog layer. Then, we use this baseline to optimize our solution.

6. ACKNOWLEDGMENTS

This work was partly supported by the German Federal Ministry for Economic Affairs and Energy (BMWi) through the KI-SIGS – KI-Space for intelligent health systems (grant no. 01MK20012P). Furthermore, we thank Holmer Hemsen for the valuable input and discussions.

7. REFERENCES

[1] A. Ahmed, H. Arkian, D. Battulga, and et al. Fog computing applications: Taxonomy and requirements. *arXiv preprint:1907.11621*, 2019.

[2] M. Akdere, U. Çetintemel, and N. Tatbul. Plan-based complex event detection across distributed sources. *VLDB Endowment*, pages 66–77, 2008.

[3] S. Akili. On the need for distributed complex event processing with multiple sinks. In *DEBS*, pages 248–249, 2019.

[4] A. Alexandrov, R. Bergmann, S. Ewen, and et al. The stratosphere platform for big data analytics. *The VLDB Journal*, pages 939–964, 2014.

[5] L. Benson, P. M. Grulich, S. Zeuch, and et al. Disco: Efficient distributed window aggregation. In *EDBT*, 2020.

[6] M. Bucchi, A. Grez, C. Riveros, and M. Ugarte. Foundations of complex event processing. *arXiv preprint:1709.05369*, 2017.

[7] J. Chen, L. Ramaswamy, D. K. Lowenthal, and et al. Comet: Decentralized complex event detection in mobile delay tolerant networks. In *IEEE*, pages 131–136, 2012.

[8] W. Fengjuan, Z. Xiaoming, and et al. The research on complex event processing method of internet of things. In *ICMTMA*, pages 1219–1222. IEEE, 2013.

[9] I. Flouris, N. Giatrakos, and et al. Issues in complex event processing: Status and prospects in the big data era. *JSS*, pages 217–236, 2017.

[10] A. S. Foundation. Apache storm, 2012. Accessed January 2020: `https://storm.apache.org/`.

[11] N. Giatrakos, E. Alevizos, A. Artikis, and et al. Complex event recognition in the big data era: a survey. *The VLDB Journal*, pages 313–352, 2020.

[12] M. Hung. Leading the iot, gartner insights on how to lead in a connected world. *Gartner Research*, pages 1–29, 2017.

[13] I. Kolchinsky and A. Schuster. Join query optimization techniques for complex event processing applications. *VLDB*, pages 1332–1345, 2018.

[14] I. Kolchinsky and A. Schuster. Real-time multi-pattern detection over event streams. In *MOD*, pages 589–606. ACM, 2019.

[15] J. Lin, W. Yu, N. Zhang, and et al. A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications. *IoT*, pages 1125–1142, 2017.

[16] M. P. Madumal and et al. Adaptive event tree-based hybrid cep computational model for fog computing architecture. In *ICTer*. IEEE, 2016.

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

[18] N. W. Paton and O. Díaz. Active database systems. *ACM CSUR*, pages 63–103, 1999.

[19] N. P. Schultz-Møller, M. Migliavacca, and P. Pietzuch. Distributed complex event processing with query rewriting. In *DEBS*, pages 1–12, 2009.

[20] E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In *SIGMOD*, pages 407–418, 2006.

[21] M. Zaharia, R. S. Xin, P. Wendell, and et al. Apache spark: a unified engine for big data processing. *ACM*, pages 56–65, 2016.

[22] S. Zeuch, A. Chaudhary, B. Del Monte, and et al. The nebulastream platform: Data and application management for the internet of things. *CIDER*, 2020.

[23] H. Zhang, Y. Diao, and et al. On complexity and optimization of expensive queries in complex event processing. In *SIGMOD*, pages 217–228, 2014.

[24] S. Zhang, H. T. Vo, and et al. Multi-query optimization for complex event processing in sap esp. In *ICDE*, pages 1213–1224. IEEE, 2017.