CS-E4780 Scalable Systems and Data Management Course Project
Efficient Pattern Detection over Data Streams

Submission deadline: 10.10.2025

Responsible teacher: Prof. Bo Zhao
Staff: Dr. Tuo Shi, Cong Yu, Zhongxuan Xie, Jiaxin Guo, Mustapha Abdullahi, Zheyue Tan
cs-e4780@aalto.fi

1 Background

Systems for event stream processing continuously evaluate queries over high-velocity event streams to detect user-specified patterns with low latency. Since the patterns become less important over time, it is crucial to detect them as quickly as possible. However, low-latency pattern detection is challenging, because query processing is stateful and the set of partial matches maintained by common algorithms for query evaluation grows exponentially in the size of the processed event data.

Handling intermediate states during query evaluation is particularly challenging under dynamic stream conditions. Event sources frequently exhibit variable arrival rates and diverse data distributions, which in turn lead to highly fluctuating query selectivities. During periods of elevated load, the number of events to be processed may exceed the system's computational capacity, rendering exhaustive query evaluation impractical or even infeasible. In such situations, maintaining responsiveness requires the system to apply load-shedding techniques. Rather than striving for all the results at the cost of unbounded delays, the system performs best-effort query evaluation that processes a subset of the intermediate computational results, aiming to maximize the quality of the reported results while adhering to strict latency constraints. This trade-off is fundamental in stream processing, where timely insights are often more critical than full completeness [8, 9].

2 Complex Event Processing

Complex Event Processing (CEP) is a prominent technology for robust and high-performance real-time detection of arbitrarily complex patterns in massive data streams[4]. It is widely employed in many areas where extremely large amounts of streaming data are continuously generated and need to be promptly and efficiently analyzed on-the-fly. Online finance, network security monitoring, credit card fraud detection, sensor networks, traffic monitoring, healthcare industry, and IoT applications are among the many examples.

Complex Event Processing (CEP) has become integral to modern data-driven environments because it allows organizations to identify complex temporal or semantic patterns in streaming data in real time [4]. Beyond just financial trading and fraud detection, CEP frameworks like FlinkCEP enable engineers to specify event patterns and react immediately when patterns are recognized [2]. Industry guides and glossaries emphasize that CEP matches incoming events against sophisticated patterns to recognize relationships and trigger real-time responses [5, 7]. Practical guides also highlight the architectural implications of CEP, noting that robust implementations must handle streaming data at scale, maintain low latency, and support intelligent event sourcing [7]. In short, CEP systems are not only crucial for monitoring and security but also form the backbone of modern applications that must react swiftly to complex sequences of events across many sectors [5].

The patterns detected by CEP engines are often of exceedingly high complexity and nesting level, and may include multiple complex operators and conditions on the data items. Moreover, these systems are typically required to operate under tight constraints on response time and detection precision, and to process multiple patterns and streams in parallel. Therefore, advanced algorithmic solutions and sophisticated optimizations must be utilized by CEP implementations to achieve an acceptable level of service quality.

The Figure 1 above presents an overview of OpenCEP structure[6]. Incoming data streams are analyzed on-the-fly and useful statistics and data characteristics are extracted to facilitate the optimizer in applying the aforementioned optimization techniques and maximize the performance of the evaluation mechanism – a component in charge of the actual pattern matching.

By incorporating a multitude of state-of-the-art methods and algorithms for scalable event processing, OpenCEP can adequately satisfy the requirements of modern event-driven domains and outperform existing alternatives, both in terms of the actual performance and the provided functionality.

OpenCEP features a generic and intuitive API, making it easily applicable to any domain where event-based streaming data is present.

3 Data Set: Citibike

The Citi Bike system dataset [3] provides a comprehensive record of bicycle-sharing trips in New York City, capturing millions of rides each month. The dataset is released as monthly compressed CSV files, with one row per trip, enabling fine-grained analysis of user mobility patterns. Each record includes the ride identifier, bicycle type (classic or electric), start and end timestamps, origin and destination stations (with names, identifiers, and geographic coordinates), and rider classification as either a subscriber (member) or occasional (casual) customer. When monthly data exceed one million trips, the dataset is split across multiple files within a single archive

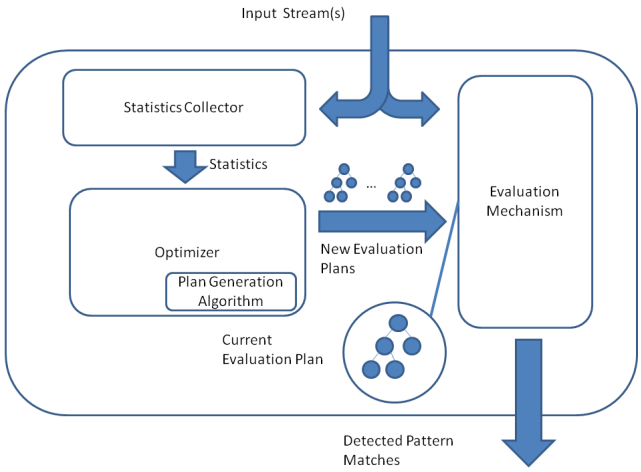

Fig. 1: Working Flow of a CEP System

to maintain accessibility. These attributes make the dataset particularly suitable for studying urban mobility, transportation demand forecasting, and the design of sustainable transportation systems.

3.1 Attributes and Format

Each trip record is structured with a consistent set of attributes: *Ride ID* (a unique trip identifier), *Rideable type* (e.g., classic or electric bike), *Started at* and *Ended at* timestamps, *Start station name* and *Start station ID*, *End station name* and *End station ID*, *Start latitude* and *Start longitude*, *End latitude* and *End longitude*, and finally, a categorical label indicating whether the trip was made by a *member* (annual subscriber) or a *casual* rider. This schema balances the need for detailed trip-level information with rider privacy, while still allowing researchers to study spatiotemporal demand dynamics, trip flows, and user behaviors.

4 Project Requirements and Problem Definition

This course project requires students to implement a basic load shedding strategy for a CEP system to handle bursty workloads while ensuring low-latency processing based on codebase OpenCEP[6]. Students will work with the Citi Bike dataset and evaluate the following query.

Query: detect hot paths. 'Hot paths' consist of stations where bicycles are accumulated faster than at other stations. They indicate the trend of movement for the bike fleet. Since bikes quickly accumulate in certain areas, the operator moves more than six thousand bicycles per day among stations. Hence, the detection of 'hot paths' of bike trips promises to improve operational efficiency.

```
PATTERN SEQ (BikeTrip+ a[], BikeTrip b)
WHERE a[i+1].bike = a[i].bike AND b.end in {7,8,9}
AND a[last].bike = b.bike AND a[i+1].start = a[i].end
WITHIN 1h
RETURN (a[1].start, a[i].end, b.end)
```

Listing 1: Citi Bike sequence pattern

Listing 1 shows a pattern detection query to detect such 'hot paths', using the SASE query language [1]: Within an hour time window, a bike is used in several subsequent trips, ending at particular stations, No. 7, No.8, and No.9 (7,8,9). Here, the Kleene closure operator detects arbitrary lengths of paths.

Project Requirement. *Students are required to implement the state management of partial matches for the **above query***, including load shedding strategies to handle bursty workloads while ensuring low-latency processing. The implementation should be based on the OpenCEP codebase [6]. The measure of success will be the ability to process incoming events with (1) low latency, even under high load conditions, while (2) maintaining the recall of pattern detection.

4.1 Definitions

We summarize key Complex Event Processing (CEP) concepts used in this project and needed to reason about the query in Listing 1 and the required load shedding logic:

Primitive (raw) event A single input record arriving on a stream (e.g., one bike trip). It has a unique timestamp and a set of typed attributes.

Attribute Named field inside an event (e.g., start station id, end station id, start time).

Event time The timestamp embedded in the event payload describing when the underlying real-world action occurred.

Processing time The wall-clock time at which the CEP engine observes/processes the event. Event time and processing time may differ under delay or disorder.

Complex event / Match A higher-level event produced when a set (typically an ordered sequence) of primitive events satisfies a pattern.

Pattern A declarative specification (here SASE) describing structural (ordering, repetition), temporal (window), and predicate constraints over events.

Sequence (SEQ) An operator requiring a temporal / positional order of constituent events.

Kleene closure (+) Operator allowing one-or-more repetitions of a sub-pattern, creating potentially unbounded numbers of partial matches.

Partial match / Partial state Data structure holding the events selected so far for a pattern instance that is not yet complete. Managing the population of partial matches dominates memory and CPU cost.

State store Logical repository (in-memory tables, lists, indexes) the engine uses to retain partial matches and auxiliary metadata during evaluation.

Correlation predicate A condition relating attributes across different events in a partial match (e.g., same bike id, station chaining a[i+1].start = a[i].end).

Selection predicate / Filter A local condition on a single event (e.g., b.end in 7,8,9).

Window (WITHIN T) Temporal constraint limiting the maximal span (end time minus first event time) of a match; enables pruning of stale partial matches.

Selectivity Fraction of incoming events (or event combinations) that survive predicates or advance partial matches; impacts growth of state.

Load shedding Intentional dropping or early termination of processing for some events or partial matches under resource pressure to keep latency bounded.

Shedding unit The granularity at which the system discards work: event-level (skip ingest), partial-match-level (prune oldest / lowest-utility states), or predicate-level (relax a constraint temporarily).

Utility / Score Heuristic value estimating the expected contribution of a partial match/event toward producing a future complete match (e.g., based on remaining window time, path length so far, station popularity).

Latency Time between arrival (or event time) of the last contributing primitive event and emission of the complex event result.

Throughput Number of primitive events processed per second.

Recall Fraction of true pattern matches still reported after any shedding (measures correctness under best-effort mode).

Pruning Safe elimination of partial matches proven impossible to complete (e.g., window expired, violated chaining constraint) without loss of recall. Note this is distinct from load shedding.

Expiration Event or partial match removal triggered by window boundaries or watermarks.

Policy examples to shed (i) Probabilistic sampling of new partial matchs/events, (ii) Lowest-utility partial matchs/events drop, etc.

For this project, focus on: (1) measure the overload detection latency and throughput under bursty workloads, (2) implement a load shedding strategy that balances latency and recall under bursts (e.g., prioritizing longer chains nearing completion at target stations 7,8,9 within 1h), and (3) evaluate the effectiveness of the shedding strategy in maintaining low latency while maximizing recall.

5 Submission Requirements

Students must implement a system to solve the problems described in Section 4. Each student is required to submit a link to the system's GitHub repository (ensure the repository is set to public visibility) along with a 4 page report detailing the system. The detailed requirements for the project report are as follows:

5.1 Project Report Requirements

Template. Use the ACM Proceeding Template: LaTeX[1] or Word [2].

Report Organization. The report should be 4 pages and include the following sections. Students may use their own section titles, but the content should align with the following structure:

(1) **Abstract.**
(2) **Introduction.** Provide a brief background and overview of the project.
(3) **System Architecture.** Describe the design choices and motivations in detail.
(4) **Implementation.** Describe how the system was implemented, including the algorithmic design, key data structures, and the estimated time and space complexity of the core components.
(5) **Performance Evaluation.** Evaluate the system's performance. Measure recall under latency bounds set to 10%, 30%, 50%, 70%, 90% of the original latency without load shedding. As an advanced requirement, assess scalability—how performance varies with resources (e.g., CPU cores) and workloads (e.g., events per second)—and report resource utilization (e.g., CPU and GPU usage).
(6) **Conclusion.**

[1]https://www.overleaf.com/latex/templates/acm-hypertext-conference-template/pchbkqfnmxgr
[2]https://www.acm.org/binaries/content/assets/publications/word_style/interim-template-style/interim-layout.docx

References

[1] Jagrati Agrawal, Yanlei Diao, Daniel Gyllstrom, and Neil Immerman. 2008. Efficient pattern matching over event streams. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. 147–160.

[2] Apache Software Foundation. 2025. *FlinkCEP-Complex event processing for Flink*. https://nightlies.apache.org/flink/flink-docs-master/docs/libs/cep/ Documentation for an unreleased version of Apache Flink.

[3] Citi Bike. 2025. Citi Bike System Data. https://citibikenyc.com/system-data.

[4] Gianpaolo Cugola and Alessandro Margara. 2012. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys (CSUR)* 44, 3 (2012), 1–62.

[5] Databricks. 2024. *What is Complex Event Processing [CEP]?* https://www.databricks.com/glossary/complex-event-processing

[6] Ilya Kolchinsky. 2025. OpenCEP: Complex Event Processing Engine. https://github.com/ilya-kolchinsky/OpenCEP. Accessed: 2025-08-25.

[7] Redpanda Data. 2024. *Complex event processing—Architecture and other practical considerations.* https://www.redpanda.com/guides/event-stream-processing-complex-event-processing

[8] Cong Yu, Tuo Shi, Matthias Weidlich, and Bo Zhao. 2025. SHARP: Shared State Reduction for Efficient Matching of Sequential Patterns. *arXiv preprint arXiv:2507.04872* (2025).

[9] Bo Zhao, Nguyen Quoc Viet Hung, and Matthias Weidlich. 2020. Load shedding for complex event processing: Input-based and state-based techniques. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1093–1104.