CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Finding Needles in the Haystack

A CEP Approach to Detect Recurring Grid Issues

Master's thesis in Computer science and engineering

Erik Larsson

Josef Ngo

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

Master's thesis 2025

Finding Needles in the Haystack

A CEP Approach to Detect Recurring Grid Issues

Erik Larsson
Josef Ngo

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2025

Finding Needles in the Haystack
A CEP Approach to Detect Recurring Grid Issues
Erik Larsson
Josef Ngo

Supervisor: Vincenzo Gulisano, Department of Computer Science and engineering
Advisor: Joris van Rooij, Göteborg Energi
Examiner: Vincenzo Gulisano, Department of Computer Science and engineering

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2025

Finding Needles in the Haystack
A CEP Approach to Detect Recurring Grid Issues
Erik Larsson
Josef Ngo
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The digitalization of electricity grids through the Advanced Metering Infrastructure (AMI) has led to unprecedented volumes of data and automated event generation from smart meters. While this enhanced monitoring capability provides valuable insights into grid conditions, the high frequency of generated events creates challenges for utility companies who must distinguish between routine fluctuations and genuine operational issues.

This thesis investigates the application of Complex Event Processing (CEP) to improve anomaly detection in smart meter data by correlating meter-generated events (e.g., overvoltage or physical tampering) with time-series measurement data (e.g., electricity consumption or voltage). Using real-world datasets from Göteborg Energi's AMI system three CEP-based queries were implemented in Apache Flink: A query that detects consumption anomalies following physical tampering events (terminal cover dismounted), an overvoltage query that identifies sustained voltage problems rather than momentary spikes, and a duplicate timestamp query that reveals systematic data collection issues.

The terminal cover dismounted query achieved filtering ratios between 0.46% and 1.00%, reducing 1,413 events to 6-13 actionable anomalies with 46-50% precision. The overvoltage query demonstrated filtering ratios from 0.006% to 1.44%, effectively reducing 98,547 events to manageable numbers for analyst review. The duplicate timestamp query discovered a previously unknown systematic issue affecting nearly 269,000 meters.

The queries were evaluated using throughput, CPU usage, memory usage, and latency. Performance evaluation demonstrates strong scalability characteristics with processing throughput of approximately 400,000 tuples per second, significantly exceeding the estimated production data flow of 4,000 tuples per second. The system maintained consistent CPU usage and conservative memory requirements ($\sim$10GB peak), supporting practical deployment with resources available at utility companies. Latency evaluation for the terminal cover dismounted query and the duplicate timestamp query showed a median between 21 and 50 seconds while the overvoltage query had an increased median between 230 and 270 seconds with outliers above 2600 seconds.

These findings show that CEP can enhance anomaly detection in AMI systems, enabling automated correlation of events and measurements while reducing false positives and analyst workload. The approach shows promise for grid monitoring applications and provides a foundation for more sophisticated anomaly detection

systems using CEP. Future research can investigate ways to further minimize latency, enabling operators to detect and respond to anomalies more promptly.

Keywords: Complex Event Processing, Stream Processing, Smart Grid, Advanced Metering Infrastructure, Anomaly Detection, Apache Flink

Acknowledgements

We would like to extend our utmost gratitude to our advisor, Joris van Rooij, and the entire team at Göteborg Energi for providing us with this opportunity and the resources throughout our thesis. We are especially grateful for our supervisor, Vincenzo Gulisano, for his guidance and support throughout the semester.

Erik Larsson, Josef Ngo, Gothenburg, 2025-08-07

Contents

List of Figures

List of Tables

1

Introduction

The digitalization of electricity grids has led to vast amounts of data being generated by smart meters and sensors in electricity grids [1], [2]. This data captures a wide range of system parameters, forming time series such as energy consumption, voltage and frequency. While most of this data reflects normal operation, certain patterns indicate anomalies that require attention to prevent system failures.

Identifying these anomalies manually is inefficient and impractical as it would entail inspecting hourly data from hundreds of thousands of meters, an infeasible task without automation. It can also require an expert in electric metering data such as an analyst or grid operator to determine what constitutes an anomaly. As such, there is a need for automated and continuous real-time analysis with timely notifications when an anomaly is detected [3].

Recent upgrades to Gothenburg's metering infrastructure, operated by Göteborg Energi (GE), have introduced higher data granularity and advanced event generation. Each meter now outputs two types of data, (1) batches of time series data containing measurements (e.g., voltage, consumption) transmitted at regular intervals, and (2) real-time meter events, sent immediately when conditions like overvoltage or potential tampering are detected.

While these events could be used to alert operators about important occurrences, the frequency of generated events is high, and many do not necessarily indicate real issues. Individual meter events may capture short-term fluctuations that are not operationally significant, leading to unnecessary alerts.

To improve decision-making, there is a need to detect higher level event patterns (anomalies) that are more strongly correlated with actual grid problems. These higher level events are easier for grid operators to interpret and can more reliably signify important occurrences in the system.

Previous work has demonstrated that stream processing and Complex Event Processing (CEP) are effective approaches for managing the scale and complexity of smart meter data [4], [2]. These techniques support real-time monitoring and high level interpretation of meter data. While stream processing focuses on handling continuous flows of data efficiently, CEP provides a higher level abstraction for detecting patterns, correlations and sequences of events across time. They have previously been applied to infer higher level events based on meter series data, such as anomalies that indicate faulty installations and tampering [5] or dangerous voltage levels [3].

With the emergence of meter generated events, some anomalies are now detected and reported at the edge of the grid, rather than inferred through central analysis. While this represents a step toward more advanced analysis and transparency in grid operation, there are still improvements to be made. This work builds on established CEP techniques, taking a novel approach by combining these meter events with series data (modeled as a data stream) to explore the potential of further identifying more meaningful and actionable patterns.

1.1 Aim

The aim of this thesis is to explore how CEP can be used to improve anomaly detection in the Gothenburg electricity grid by refining the interpretation of smart meter data. This includes:

- Developing methods to identify higher level event patterns that are more strongly correlated with actual grid issues, reducing the reliance on frequent low-level meter events that may not indicate real problems.
- Investigating how the combination of the meter's event stream and series data stream can be leveraged to filter out false positives, ensuring that alerts are triggered only for meaningful anomalies.
- Evaluating the effectiveness, scalability, and operational feasibility of CEP in improving anomaly detection, with a focus on reducing unnecessary interventions while maintaining sensitivity to critical grid conditions.

This work aims to enhance the reliability of real-time grid monitoring by ensuring that the detected anomalies align more closely with actual operational concerns.

1.2 Limitations

Domain-specific focus. The study will be tailored to smart meter data within Gothenburg's electricity grid. While the methodology may be applicable to other grids, generalization to different infrastructures or industries is beyond the scope of this work.

Existing technology stack. The implementation will be based on Apache Flink and the infrastructure currently deployed at GE. The evaluation will focus on this specific setup. Alternative streaming frameworks will not be considered.

2

Technical Background

This chapter provides an overview of the technical concepts and technologies relevant to this work, detailing Advanced Metering Infrastructure, Stream processing and Complex Event Processing.

2.1 Advanced Metering Infrastructure

Advanced Metering Infrastructure (AMI) allows utilities to monitor energy consumption more efficiently through smart meters, which measure electricity usage and communicate data to central servers. The primary goal of AMI is to automate meter readings, enabling utility companies to collect data remotely. This is in contrast to before the introduction of AMI where the consumption data was, infrequently, read manually on-site by personnel [4].

In Sweden, AMI was widely deployed starting in 2009, driven by the need for monthly meter readings [4]. Over time, the technology has evolved. Previous regulations required hourly reports, but by this year (2025), all smart meters are required to record data with a 15 minute granularity, including measurements for active and reactive energy, voltage, current, and power [6]. This increase in data collection enables a more detailed view of electricity usage and grid conditions.

A key component of this infrastructure in the Gothenburg grid is the *Kamstrup OMNIA e-meter*, operated by GE. Apart from recording measurements, these smart meters also support event generation [7]. Event types include basic notifications like voltage thresholds being exceeded whereas other events can point to broader disturbances in the grid. The improved infrastructure not only increases the granularity of data but also enhances event detection capabilities.

As a result of these advancements, the volume of data collected has grown significantly. While this data contains valuable information, not all of it is immediately relevant. The large data flow presents a challenge: efficiently extracting meaningful insights without being overwhelmed by irrelevant information. To support real-time analysis of such data flows, stream processing can be used to continuously manage and organize the data before applying methods like CEP.

2.2 Stream processing

A stream S is defined as an unbounded sequence of tuples t generated continuously in time [8], formally $S = \langle t_1, t_2, t_3 ... \rangle$. The tuples share a common schema of attributes, thus, a tuple t can be defined as $t = \langle a_1, a_2, ..., a_n \rangle$. Figure 2.1 exemplifies how tuples could be structured for AMI data.

<Timestamp, Meter ID, Consumption>

<12:00, MID-1, 72>
<12:00, MID-2, 38>
<13:00, MID-1, 71>

Figure 2.1: Example of a Tuple schema for AMI data containing timestamps, meter IDs, and electricity consumption readings.

Stream processing is a computational paradigm where operators are continuously applied on a stream, in contrast to batch processing where data is collected, stored and then processed in batch intervals. The streaming approach is particularly advantageous in modern data processing scenarios where data is generated continuously, such as web logs, financial transactions, and data from sensors like smart meters [9]. Traditional batch-oriented systems often introduce artificial delays by grouping data into fixed intervals, leading to increased latency and inefficiencies [10]. By processing each data point upon arrival, stream processing reduces computational overhead, minimizes storage requirements through immediate filtering and incremental aggregation. Furthermore, it enables timely output of the computation, allowing real-time decision-making that can be crucial for use cases like electric grid monitoring.

2.2.1 Streaming Processing Engines

The stream processing paradigm is implemented through frameworks known as Stream Processing Engines (SPEs), which are designed to process continuous data streams in parallel across multiple nodes in a distributed system. One such framework is Apache Flink (Flink), which originated as a research project before becoming an Apache project in 2014 and is currently used in production by several companies. It is used for proofs-of-concepts at GE.

2.2.2 Event Time

In stream processing, event time refers to the actual moment when an event was generated at its source, rather than when it was processed by the system. Each tuple contains an embedded timestamp that represents when the event truly occurred. For example, in AMI systems, this would be the moment when a smart meter recorded a consumption value or generated a meter event. In both cases, the timestamp is referred to as event time. Unlike processing-time approaches which depend on system clocks, event time processing relies on these data-embedded timestamps. This temporal decoupling creates a fundamental challenge: the system must determine

when it has received all events up to a certain event time before computing results. This challenge is addressed through watermarking (explained in Section 2.2.5), a mechanism that provides stream progress indicators based on observed timestamps.

2.2.3 Operators

Operators can be categorized based on how they process input and produce output. Some operators, such as map, takes one input tuple and produces exactly one output tuple without maintaining any state. Others, like flatMap or windowed aggregations, may produce zero, one, or multiple output tuples per input and often rely on stateful processing to maintain context across tuples. If an operator combines multiple input tuples for a single output, this is referred to as a stateful operator, which means it needs to wait for all of the tuples to arrive before processing them.

The relevant operators for this thesis are the stateless *Filter* and the stateful *Aggregate* and *Join*.

- **Filter** Outputs tuples that match a boolean condition.
- **Aggregate** Outputs tuples, where one or more input tuples are combined to one output. For example, a calculated average over a time period.
- **Join** Outputs a combination of two different input streams. For example, an output tuple can consist of a measurement from the first stream, and an event from the other.

Before applying stateful operations, it is often necessary to partition the stream to ensure that only related data is processed together. For example, when computing an average value for a given meter, only tuples originating from that specific meter should be included in the calculation. This can be achieved using the *KeyBy* function, which partitions the stream based on a specified key, in this case the meter ID.

2.2.4 Windows

Stateful operators require a mechanism to determine when to collect tuples and when to finally execute their intended task or calculation. This could be after a certain amount of input tuples, however, it is often more useful to base it on time. Using time allows processing to be more aligned with real-world events and can provide more timely results, rather than waiting for a fixed number of tuples which may arrive at irregular intervals or not at all. The key concept that enables this is windows.

Windowing is a technique to split an unbounded stream of tuples into finite chunks, allowing operations to be performed on the contents within the window. Formally, a window W over a stream S is a finite subset of S [8]. There are a few common types of windows, each serving different purposes. For this thesis, the relevant window type will be tumbling windows. They are fixed-size, non-overlapping windows. Each window spans a predefined time interval and a tuple belongs to exactly one window.

When a window closes, a new one opens, ensuring that all tuples are processed exactly once within a given time frame, as seen in Figure 2.2.

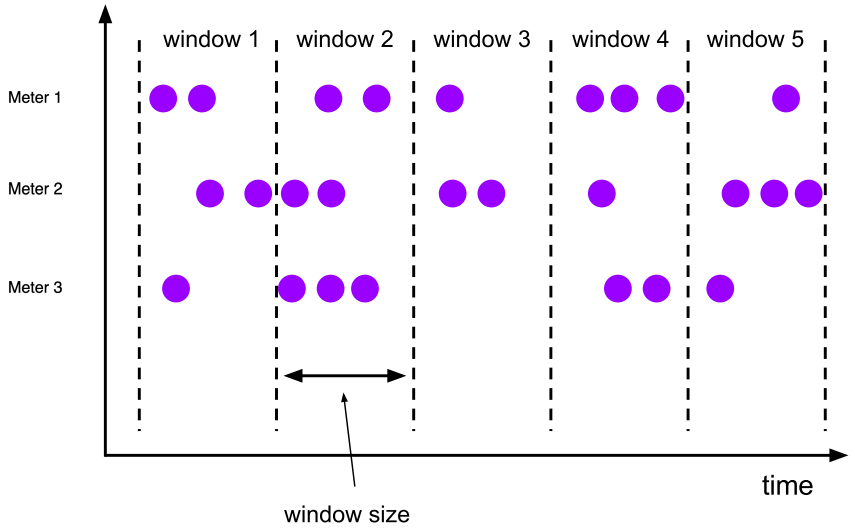

Figure 2.2: An illustration of tumbling windows, showing that this window type does not overlap. Adapted from the Flink documentation [11] under Apache License 2.0.[1]

The size of windows is typically defined based on time. However, challenges such as network delays and out-of-order events complicate the task of determining when all relevant data has arrived [13]. Latency can cause some data to arrive late, regardless of whether the window closes after a fixed duration. If the system relies solely on the timestamps embedded in the tuples for window progression, out-of-order events can lead to indefinite delays in processing. This is because the system may continue to wait for the late events that belong to a specific time frame, preventing it from closing the window and computing results. To address these challenges, Flink employs watermarks, which is discussed in the following section.

2.2.5 Watermarks

To deal with the task of determining when a system has received all events within a specific time window, modern stream processing frameworks implement watermarking mechanisms that track event time progression through the data stream itself [13]. Watermarks function as special markers embedded within the data stream, carrying a timestamp value that indicates temporal completeness up to that point. Formally, when a watermark with timestamp τ passes through the system, it asserts that all

[1]This thesis includes materials (e.g., diagrams and images) originally from official Apache project documentation, which is licensed under the Apache license, Version 2.0. You may obtain a copy of the license at [12]. Modifications have been indicated where applicable. All such usage complies with the terms of the Apache License, Version 2.0.

events with timestamps less than or equal to τ have already been observed (or can be considered as not arriving). This signal enables time-based operations such as windows to be informed about when calculations can be made.

The watermark propagation allows operators to advance their internal event-time clocks accordingly. For example, when an hourly window operator receives a watermark with timestamp 14:00, it can safely conclude that the 13:00-14:00 window is complete and can be processed for results, as no more events belonging to this window are expected.

The effectiveness of watermarks depends on the accuracy of the timestamps and the watermark generation strategy. An overly aggressive watermark strategy might prematurely close windows and exclude late events, while an excessively conservative approach introduces unnecessary processing delays [9]. Since systems cannot wait indefinitely for late-arriving data, some maximum delay threshold must be established. This creates a trade-off between result completeness and processing latency.

2.2.6 Out-of-orderness and lateness

A common watermarking strategy is bounded out-of-orderness, which works by placing an upper bound on how far the events may arrive out of their natural event time order. Events arriving out of order frequently occur in distributed systems due to network delays, processing paths, or data sources with varying latencies [14].

Essentially this follows the same rule as event time processing with an offset. Given an out-of-order bound B and a timestamp τ, no events older than timestamp $\tau - B$ are expected to arrive after the current watermark. This watermark strategy introduces a delay of B in processing completeness, as the watermark lags behind the latest observed event to accommodate late arrivals [15].

The selection of bound B represents the trade-off mentioned above. Where a larger B increases result completeness by accommodating more late events, but at the cost of increased processing latency. Conversely, a smaller B reduces processing latency but may lead to more dropped events.

Late events are those arriving after the watermark has advanced beyond their timestamp, including the configured out-of-orderness bound. These tuples are dropped in instances where a stateful operator triggers, e.g., an event-time window is closed, and they are consequentially not part of computation within that window. They can be included in a secondary computation by specifying an allowed lateness parameter that keeps the state of the window after it is closed and then includes tuples that are marked as late but still within the configured allowed lateness. This would result in two distinct outputs from a single window, which can be useful when needing both a timely computation followed by a later more accurate computation [16].

2.2.7 Interval Join

In this thesis interval joins are essential to connect the meter events with the series data for further processing. The interval join connects tuples from two streams

(referred to as streams A and B) based on two criteria: a shared key and a temporal relationship between their timestamps.

For tuples to be joined, the timestamp of a tuple from stream B must fall within a specific time window relative to the timestamp of a tuple from stream A. This temporal relationship can be formally defined as:

$$a.timestamp + lowerBound \leq b.timestamp \leq a.timestamp + upperBound$$

Where tuples a and b (from streams A and B respectively) share a common key. The bounds can take either positive or negative values, with the only constraint being that the lower bound cannot exceed the upper bound.

To illustrate the concept, consider the example where two streams orange and green are joined with temporal bounds of -2 milliseconds (lower) and $+1$ millisecond (upper). By default, these boundaries are inclusive. The triangles in Figure 2.3 illustrate which tuples from the orange stream are joined with in the green stream.

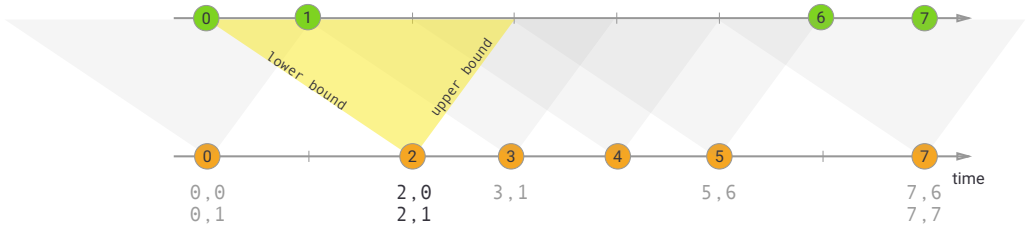

Figure 2.3: Visualization of interval join with lower bound of -2ms and upper bound of $+1$ms. From the Flink documentation [17] under Apache License 2.0.

2.3 Complex event processing

Traditional stream processing systems focus on ingesting, transforming, and analyzing continuous data streams. CEP extends this paradigm by enabling the detection of meaningful patterns, relationships and trends across multiple lower level events. CEP utilizes a higher level abstraction to express rules or queries that specify patterns of interests [18], allowing users to create semantically rich applications through a user-friendly API. While CEP offers improved usability, it has been established that any CEP query can be expressed through aggregates in traditional stream processing [19].

In this work, the Flink library FlinkCEP is used and the descriptions, capabilities and usages of CEP discussed in the thesis are from the perspective of this library.

2.3.1 FlinkCEP

FlinkCEP is a library built on top of Flink that allows detection of complex patterns in event streams using a high level declarative API [20]. The library processes events in event-time order and rely on watermarking to handle out-of-order data. Incoming events are buffered and sorted by timestamp before being evaluated against defined patterns.

Patterns are defined as sequences of named sub-patterns with associated conditions. Let A, B, and C denote individual sub-patterns and P the full pattern. FlinkCEP provides several operators to define how these patterns relate to each other and what constitutes a match.

Ordering between pattern tuples: The `.followedBy()` (denoted $\rightarrow$) operator enforces order with relaxed contiguity (i.e., other events may appear in between), and the `.next()` (denoted $\Rightarrow$) operator enforces order with strict contiguity (i.e., one event must follow another without any intervening events).

Pattern definition: `P = {A` $\rightarrow$ `B` $\rightarrow$ `C}`
This requires `A`, `B`, and `C` to appear in this sequence, but other events can occur in between.

- *Matches:* `[A, B, C]`, `[A, X, B, Y, C]` (`X` and `Y` occur between, but the order of the pattern is preserved)
- *Non-matches:* `[A, C, B]` (no `C` follows `B`)

Pattern definition: `P = {A` $\Rightarrow$ `B` $\Rightarrow$ `C}`
This requires `A`, `B`, and `C` to appear in exact sequence with no events in between.

- *Matches:* `[A, B, C]`
- *Non-matches:* `[A, C, B, C]`, `[A, C, B]` (any interruption breaks the match)

Repetition: The `.times(n)` and `.timesOrMore(n)` operators control how many times a sub-pattern must appear.

Pattern definition: `P = {A` $\rightarrow$ `B.times(2)}`
This pattern matches sequences where `A` is followed by two occurrences of `B`.

- *Matches:* `[A, B, B]`, `[A, B, C, B]` (the `B`s are separated by a `C` but appear twice after `A`)
- *Non-matches:* `[A, B, C]` (only one `B`), `[B, A, B, C]` (`B` appears before `A`)

Pattern definition: `P = {A` $\Rightarrow$ `B.times(2)}`
This pattern matches sequences where `A` is directly followed by two occurrences of `B`.

- *Matches:* `[A, B, B]`,
- *Non-matches:*`[A, C, B, B]` (the `C` stops the match)

Pattern definition: `P = {A` $\rightarrow$ `B.timesOrMore(1)}`
This pattern matches sequences where `A` is followed by at least one occurrence of `B`.

- *Matches:* `[A, B]`, `[A, B, B]`, `[A, C, B]`
- *Non-matches:* `[A, C]` (no `B`)

Pattern definition: `P = {A ⇒ B.timesOrMore(1)}`
This pattern matches sequences where `A` is directly followed by at least one occurrence of `B`.

- *Matches:* `[A, B]`, `[A, B, B]`
- *Non-matches:* `[A, C, B]` (`B` is seen at least once, but not directly after `A`)

Strict Ordering within a sub-pattern: The `.consecutive()` operator enforces that repeated tuples occur without interruption.

Pattern definition: `P = {A → B.times(2).consecutive()}`
This pattern matches sequences where `A` is followed by two consecutive occurrences of `B`.

- *Matches:* `[A, B, B]`, `[A, C, B, B]`
- *Non-matches:* `[A, B, C, B]` (`C` breaks the consecutive `B`s)

By combining these operators, complex event patterns can be precisely defined and used to detect meaningful sequences in data streams.

3

Problem Definition

This chapter defines the problem addressed through the work done in this thesis by describing the system context, the specific issues in focus and the research questions.

3.1 System overview

The system considered in this thesis consists of the AMI hardware as described in Section 2.1. The smart meters measure 19 different series but in this work only four are used, the active energy and the voltage for all three phases. Averages over 15 minutes are stored in the meter's memory and are sent every second hour. The meter events used in this analysis are the 'Terminal Cover Dismounted' event and the 'Overvoltage' event, which are sent as soon as they are generated on the smart meters. Note that in this thesis, both measurements and meter events will be referred to as tuples with different schemas. This data is received by central servers, and is then made available to internal software systems at GE.

While this currently adds several steps and therefore latency, the end goal is that applications should have near real-time access to the streams of data. For now a dataset has been stored for analysis to allow reproducibility in the tests. The prototypes developed for this thesis are applied to this dataset. However, they are designed to emulate the future application of real-time analysis on ephemeral and unbounded data streams.

3.1.1 Tuple overview

The dataset is modeled as two distinct data streams, one for series data and one for meter event data. Series data has one tuple schema and meter events has another, defined in the tables below:

Field	Value	Explanation
series_id	Voltage phase 1	The specific data series, which represents what kind of measurement data is received e.g., voltage or power consumption.
meter_id	9876543219876543	Unique identifier for the specific meter.
time	2025-01-01 12:34:56.7	Timestamp of when the measurement was recorded.
value	34.567	The actual measurement that was recorded.

Table 3.1: Example of a Series tuple

Field	Value	Explanation
event_name	Overvoltage	The specific event that the meter has detected e.g., Overvoltage or Terminal cover dismounted.
meter_id	9876543219876543	Unique identifier for the specific meter.
time	2025-01-01 12:34:56.7	Timestamp of when the event was generated.

Table 3.2: Example of a Meter Event tuple

3.2 System problem

The increased granularity of smart meter data and the introduction of automated meter event generation have significantly improved grid monitoring capabilities for analysts at GE. However, the volume of generated events has made it infeasible for them to manually investigate each anomaly. Many of these events do not necessarily indicate real problems, yet they still require expert evaluation to determine their significance. Due to technical limitations and time constraints, experts at GE cannot systematically analyze every event, even though they are aware that some may reveal critical issues within the grid.

To address this, GE has identified a set of recurring problems that could benefit from a more efficient detection approach. Among these, two cases have been selected for further exploration in this thesis. Additionally, a previously unknown problem that was reported during this work is also explored.

Terminal cover dismounted event This event is triggered when a meter's enclosure is opened, which could indicate routine maintenance or unauthorized tampering. This is because only GE employees are allowed to open it, and they do so in cases where maintenance is needed. During maintenance, there have been instances of incorrect installations such as mounting cables in the wrong sequence (explored in [5]). Currently, there is no automated way to verify whether a lid opening has resulted in unintended consequences. By analyzing the average power consumption before and

after the event, it may be possible to detect incorrect installation or maintenance errors by GE personnel. Additionally, if the consumption drops unexpectedly following a lid opening, this could suggest energy theft, where power is being diverted away from the meter after unauthorized tampering.

Overvoltage event The smart meters are supposed to operate around 230 volts. They generate this event as soon as the voltage surpasses a predefined threshold, currently configured to 245 volts. This is the most common event type in the system, and consequently the most noisy. To differentiate between common minor fluctuations and real grid problems, it is necessary to consider the severity and frequency of overvoltage events. Specifically, an event is more concerning if the recorded average voltage is unusually high around the time the meter generates the event. This would mean it is not a sudden peak, but rather a persisting issue. Identifying such anomalies may point to malfunctioning smart meters. The high voltage levels can lead to inaccurate billing, damage to equipment or injury to persons [3].

Duplicate timestamps In some cases, meters may transmit multiple readings with identical timestamps for data in the same series. This behavior could indicate faulty meter behavior or problems with the data collection systems and should prompt further investigation. This problem was not previously identified but came up during the thesis work. While the implications surrounding this issue are still not perfectly clear, preliminary analysis shows that there are possibly multiple causes that can be detected by analyzing the results of such a query. Identifying this anomaly does not entail a meter generated event.

The challenge is not only detecting these anomalies but doing so in a way that reduces false positives while ensuring that actual problems are identified and prioritized. Current methods do not provide an efficient way to combine and analyze meter event streams with raw measurement data in real-time. This thesis will investigate if CEP can bridge this gap by enabling automated correlation of events and measurements, helping GE focus on anomalies that are most likely to require action.

3.3 Research questions

- How can CEP be applied to find electric grid anomalies using smart meter measurements and event data?
- What are the scalability characteristics (latency, throughput, memory and CPU usage) of such a CEP system?

3.4 Evaluation metrics

To assess the effectiveness of the CEP system, a set of evaluation metrics is defined. The goal of the system is to ensure that the data analyst is alerted on actual grid problems essentially acting like a filter on the incoming event and data stream.

One way to evaluate such a system is by comparing the amount of events generated by the smart meters and the amount of anomalies generated by the CEP system: $\frac{CEP\ anomalies}{Meter\ events}$. This metric can determine how well the system acts as a filter on the incoming event and data stream. This ratio can also be used to compare different configurations for the CEP system.

The anomalies generated by the CEP system can then be further verified by an independent expert data analyst within the domain to confirm that an anomaly correctly corresponds to something of interest, warranting manual inspection. This can be expressed as: $\frac{Expert\ verified\ anomalies}{CEP\ anomalies}$. Such a precision metric can determine the quality and effectiveness of the CEP system's filtering mechanism, reflecting how reliably it creates alerts that are of interest to GE.

In addition to the metrics mentioned above, scalability is evaluated to determine the feasibility of the CEP system under varying workloads. The primary scalability metrics include:

- **Throughput:** Measured in tuples per second, representing the rate at which incoming meter data is processed.
- **Latency:** Measured in milliseconds (ms), representing the time between an anomaly's occurrence and its detection. This is defined as the time between the production of an anomaly and the arrival of the latest tuple contributing to that output.
- **Memory:** Measured in MB, representing memory usage of the Flink application.
- **CPU Utilization:** Measured in %, representing the CPU usage of the Flink application.

4

Design and Implementation

This chapter details the design and implementation of the anomaly detection queries used to detect the different types of anomalies specified in the previous chapter.

4.1 Implementation

In this section, descriptions of how anomalies are detected using Flink will be provided.

It should be noted that the reason for not implementing a pure CEP application is that FlinkCEP does not provide a native way to express a query over two data streams. In other words, a stream processing operator is required to join together two streams into one before a CEP pattern can be applied to data from both streams.

The Flink queries uses stream processing in conjunction with FlinkCEP, where all streams use a *keyBy* based on the meter ID unless otherwise stated. The diagrams supporting the descriptions of these queries use the following symbols to visualize the flow of tuples through the different operators:

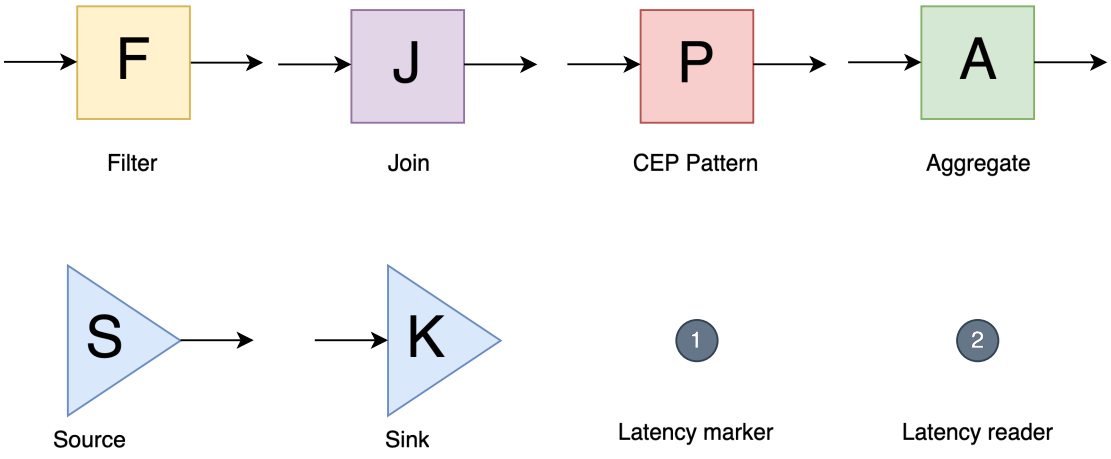

Figure 4.1: How queries will be illustrated below.

4.1.1 Terminal cover dismounted query

The first step involves processing power consumption data by filtering out the consumption series. The filtered tuples are segmented into one-hour tumbling event time windows. Within each window, the system calculates the average consumption using a custom aggregator named BucketAverage. This aggregator keeps track of the average consumption based on the data seen so far, where the buckets are keyed by the meter ID and the hour of the day (e.g., 12:00). This average represents an expected value for a specific meter on a specific hour of day. This is due to the fact that electricity consumption usually follows a pattern each day [21]. The output is an enriched average tuple, containing the average for a given hour as well as the expected consumption for that hour.

In parallel with the consumption data, a second stream of meter events is ingested. This stream is filtered to extract events that indicate physical tampering, specifically the event called 'Terminal cover dismounted'. These tuples are used for later correlation with the corresponding consumption data for a given meter.

To associate potential tampering events with subsequent changes in consumption, an interval join is performed between the enriched average consumption stream and the filtered event stream. This join matches each event tuple with all consumption tuples from the same meter that occur within a five-hour window following the event.

The result is a combined data stream where each resulting tuple links a consumption observation with a preceding tampering event. The final anomaly detection logic is implemented using a FlinkCEP pattern.

Pattern Definition: `P = {A.times(3).consecutive()}`
This pattern is defined to detect at least three consecutive tuples where the observed consumption is below the expected level multiplied by a certain threshold. The subpattern `A` uses the following calculation as a condition for filtering incoming tuples.

Threshold Calculation:
Let `O` denote the observed value, `E` the expected value, and `T` the threshold percentage. The condition for a tuple to be emitted is:

$$O < E \times T$$

For example, if the observed value is `O = 8` with an expected value of `E = 15` and a threshold of `T = 60%`, the condition becomes:

$$8 < 15 \times 0.6 \implies 8 < 9$$

As this condition is satisfied, the tuple will be emitted as it is below 9 (15×0.6). The pattern is then applied with repetition and strict contiguity where single tuples will not be emitted to the sink, only instances where three tuples in a row are emitted by `A` are finally emitted by `P`. Thus, tuples emitted from the resulting stream serve as alerts that can notify analysts of potential anomalies in consumption patterns.

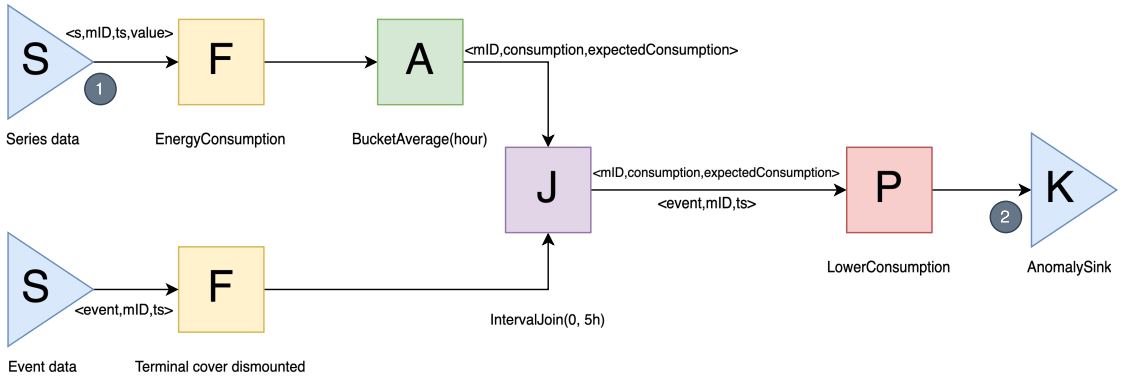

Figure 4.2: Diagram showing an overview of the terminal cover dismounted query.

4.1.2 Overvoltage query

This query filters for voltage series across all three phases, dividing the stream into three separate streams using an additional `keyBy` based on the series ID, which corresponds to the phase of the measurements.

Pattern Definition: `P = {A.times(8).consecutive()}`
The streams are processed through a FlinkCEP pattern that matches if the voltage exceeds a specified threshold for eight consecutive measurements (i.e., over a duration of two hours, since the measurements are taken every 15 minutes). This would indicate that the voltage is abnormally high over an extended period, rather than being a momentary spike. The sub-pattern `A` is defined as:

$$O > T$$

Where `O` denotes the observed value and `T` the threshold voltage. `P` emits a high voltage match when eight consecutive tuples are above the threshold.

In parallel with the voltage analysis, the stream of events is filtered by the event called 'Overvoltage, start'. This stream is then interval joined with the high voltage tuples with timestamps three hours before and after the event timestamp. Tuples emitted to the resulting stream are alerts that can notify analysts.

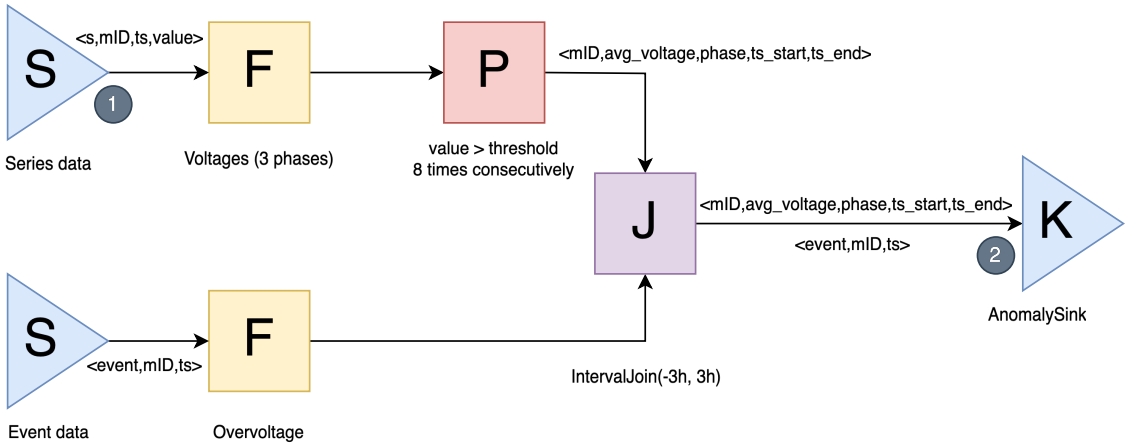

Figure 4.3: Diagram showing an overview of the overvoltage query.

4.1.3 Duplicate timestamp query

This query can process all measured series as the anomaly can potentially be detected for any of them. However, it also uses an additional `keyBy` based on the series ID as there will always be tuples with the same timestamp across different series, but there should never be two within the same series. These partitioned streams are then passed separately through a FlinkCEP pattern that compares the timestamps of a pair of tuples.

Pattern Definition: `P = {A ⇒ B}`
This pattern is defined to detect matching timestamps. It makes use of two sub-patterns, `A` and `B`, where `A` matches all incoming tuples. `B` compares the current tuple to the next tuple in the stream, which flows in from `A`, as it matches all incoming tuples. FlinkCEP ensures correctness in the temporal relationship between the tuples as they are sorted. `P` emits a duplicate match if the timestamps are identical.

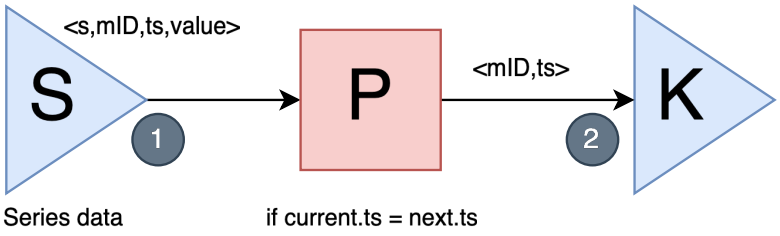

Figure 4.4: Diagram showing an overview of the duplicate timestamp query.

4.1.4 Out-of-orderness threshold

An analysis on the out-of-orderness within the series dataset was conducted to determine a tolerance threshold for the queries. The query used was designed to direct any tuple with a timestamp below the current watermark to a side output for counting. A BoundedOutOfOrderness watermark strategy with values ranging from 0 to

10 hours was evaluated. Based on these findings presented in Figure 4.5, a value of 10 hours, with a drop rate of approximately 0.7%, was selected to minimize the drop rate of tuples. As the current system does not have any real-time capabilities, result completeness was prioritized over processing timeliness while still maintaining a true emulation of the problems present in the data.

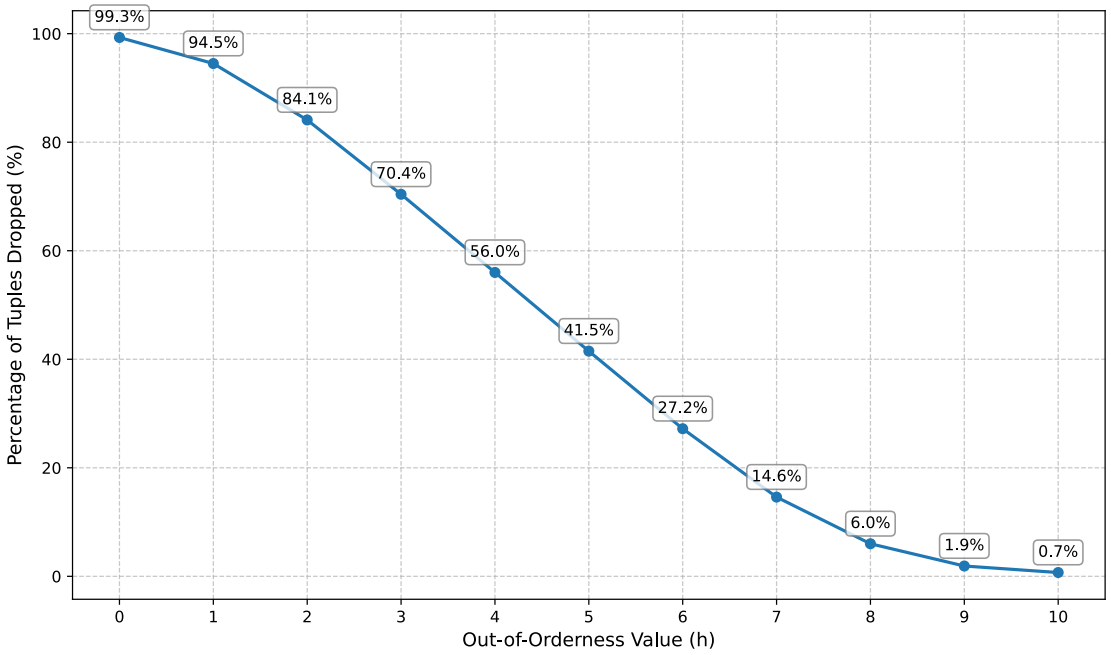

Figure 4.5: Line chart visualizing the percentage of tuples in the series data dropped with different values of bounded-out-of-orderness.

The same analysis was conducted on the event dataset. To achieve a similar rate of dropped tuples as the series dataset, an out-of-orderness value of 20 hours was required, as can be seen in Figure 4.6. Based on this observation it was decided to sort the event dataset to provide accurate results and to avoid losing 20 hours of event data, as this data is the main focus of the thesis. The implications of different out-of-orderness values are discussed in Section 6.8.

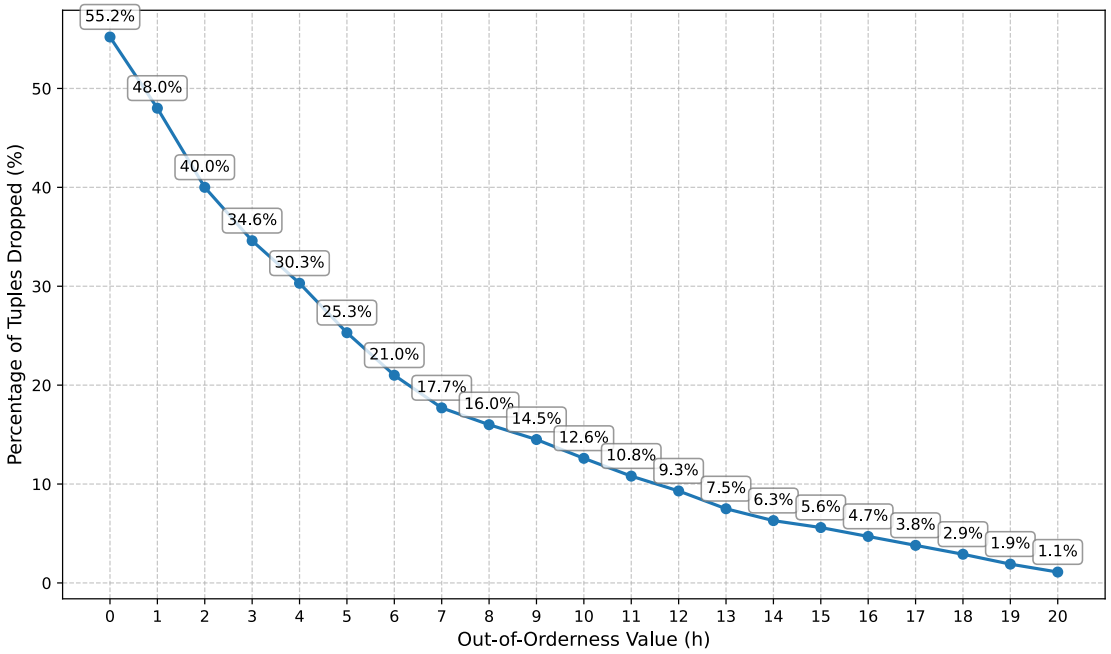

Figure 4.6: Line chart visualizing the percentage of dropped tuples in the meter event data with different values of bounded-out-of-orderness.

5

Evaluation

In this chapter the evaluation of the queries described above will be presented. The goal of the evaluation is to answer two main questions. Determine the ratio of anomalies generated by the system compared to the meter events and determine the scalability metrics of the system.

5.1 Evaluation setup

This section describes the system and dataset that were used to evaluate the queries.

5.1.1 System

The evaluations were conducted on an Azure Standard_B8as_V2 virtual machine.

The physical processor is an AMD EPYC 7763v with 8 vCPUs allocated. The virtual machine has 32GB of memory and is running the Ubuntu 24.04 LTS operating system. More information about this virtual machine is available at [22].

Flink is configured with 2GB of memory allocated to the Job Manager and 26GB of memory allocated to the Task Manager. The Task Manager uses 8 task slots, matching the number of available cores. Information about this configuration is available under Basic Setup at [23]. To make use of the available resources, all operators except the source used a parallelism degree [24] of 8.

5.1.2 Dataset

This work uses a dataset that was captured from live smart meter data in the Gothenburg grid. It was split into two datasets, with series (S) data combined with its respective meter event data (E), as specified below.

Dataset	Size (GB)	Total Tuples	Date span
S1	245	1,204,967,449	14/02/2025 - 06/04/2025 (52 days)
E1	0.36	1,712,684	26/02/2025 - 05/04/2025 (39 days)
S2	413	1,916,413,550	27/03/2025 - 01/04/2025 (6 days)
E2	0.08	372,802	27/03/2025 - 01/04/2025 (6 days)

Table 5.1: Details of the datasets used in the evaluation.

Different datasets were used depending on which query is being evaluated. Due to storage and time constraints, dataset S1 was filtered to only include the energy consumption series in order to keep the file size and query runtimes manageable with a larger set of dates. In contrast, dataset S2 includes all series data to accurately emulate the high volume of concurrent data produced by the meters, but with a significantly smaller date span as the file size grows quickly for each included day.

Dataset S1 was used for the terminal cover dismounted query and the duplicate timestamp query because it spans over a larger set of dates. The former requires historical data, as explained in Section 4.1.1. The latter benefits from running over many days as this helps reveal how widespread the issue is.

The overvoltage query uses dataset S2 since there is still a significant amount of relevant data to analyze in a shorter time span, and it does not make use of historical data. Instead when testing this query there was an opportunity to test it under the current circumstances at GE where all series flow through the same stream.

5.2 Anomaly detection results

Below the results are presented for each of the queries. The ratio is computed by comparing the amount of anomalies generated by the CEP system and the amount of events generated by the smart meters: $\frac{CEP\ anomalies}{Meter\ events}$. Similarly, precision was computed by comparing the expert verified anomalies against the amount of anomalies generated by the CEP system: $\frac{Expert\ verified\ anomalies}{CEP\ anomalies}$. Both metrics have been rounded to three significant digits.

5.2.1 Terminal cover dismounted query

The terminal cover dismounted query was tested using dataset S1 with 1,204,967,449 total series tuples and 1,712,684 total event tuples, of which 1,413 were terminal cover dismounted events. The query was evaluated at seven different thresholds: 10%, 20%, 30%, 50%, 75%, 85%, and 99%.

Threshold	Anomalies	Ratio	Precision
10%	6	0.425%	50.0%
20%	6	0.425%	50.0%
30%	10	0.708%	40.0%
50%	12	0.849%	41.7%
75%	13	0.920%	46.2%
85%	13	0.920%	46.2%
99%	13	0.920%	46.2%

Table 5.2: Results of the terminal cover dismounted query

The results show an increasing trend in anomaly detection as thresholds rise, though with some plateaus. At the lowest thresholds of 10% and 20%, the system generated

6 anomalies each (0.425% of terminal cover dismounted events). The anomaly count increased to 10 at the 30% threshold (0.708%), then to 12 at 50% (0.849%), and plateaued at 13 anomalies for the three highest thresholds of 75%, 85%, and 99% (0.920% each).

5.2.2 Overvoltage query

The overvoltage query was tested using dataset S2 with 372,802 total events, of which 98,547 were overvoltage events. The query was evaluated at four different voltage thresholds: 245V, 250V, 253V, and 255V. The results show a clear relationship

Threshold	Anomalies	Ratio	Precision
245V	1417	1.44%	N/A
250V	84	0.085%	N/A
253V	33	0.033%	100%
255V	6	0.006%	100%

Table 5.3: Results of the overvoltage query

between threshold voltage and anomaly detection frequency. At the lowest threshold of 245V, the system generated 1,417 anomalies (1.44% of all overvoltage events). As the threshold increased, anomaly counts dropped significantly: 84 anomalies (0.085%) at 250V, 33 anomalies (0.033%) at 253V, and only 6 anomalies (0.006%) at 255V.

Precision could only be calculated for the two highest thresholds (253V and 255V), both achieving 100% precision when expert-verified. The lower thresholds (245V and 250V) generated too many anomalies for the GE analyst to manually review and verify, hence the N/A precision values.

5.2.3 Duplicate timestamp query

The duplicate timestamp query was tested using dataset S1 with 1,204,967,449 total series tuples from 270,658 unique meters. Since this query does not make use of meter events, the results are visualized to show the magnitude of the anomaly. Figure 5.1 displays a histogram of the unique meters with at least one duplicate timestamp grouped by date. The top two dates contains close to 269 000 unique meters, and the following around 2000 or less.

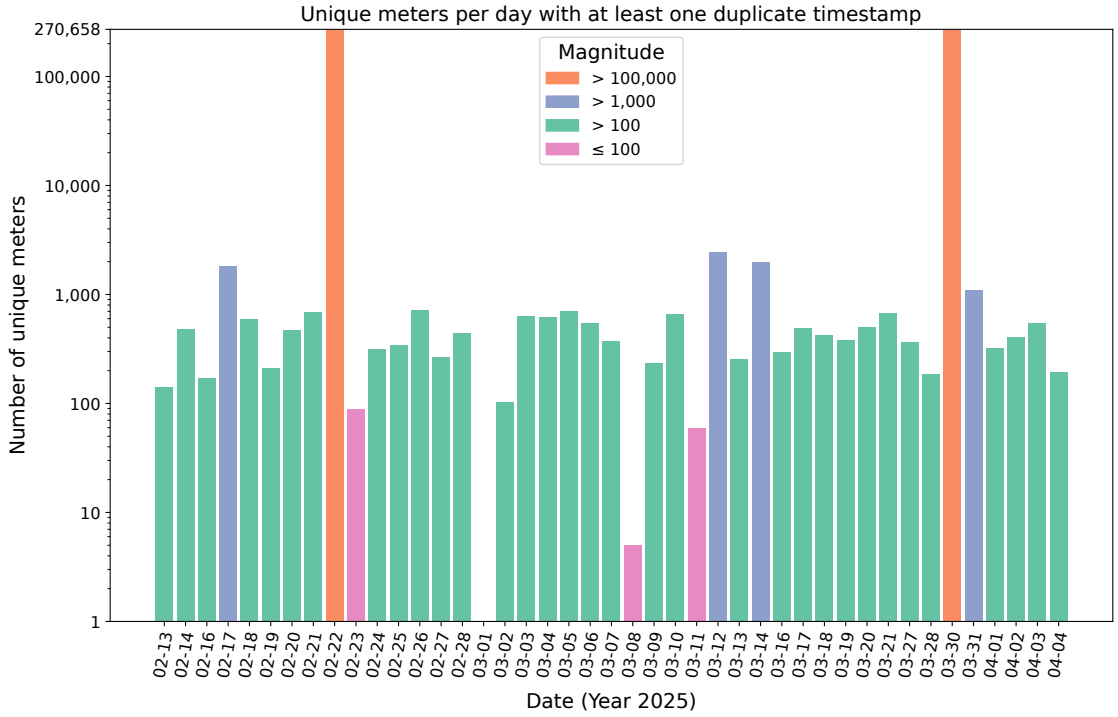

Figure 5.1: Histogram of the unique meters with at least one duplicate timestamp grouped by date.

5.3 Scalability metrics

At the start of every evaluation run, a script queries Flink's REST API for a set of metrics every ten seconds and logs this to a file. The corresponding metrics extracted from Flink are the 'numRecordsOut' (at the source) and the task manager's 'CPU Load' and 'Heap Used' [25].

Latency is extracted from the output of the queries, where the latency marker sets the start time and the latency reader sets the end time and calculates the difference (as depicted in the diagrams in Chapter 4).

To get a more accurate representation of the running state of the application, 5% of the logged values are removed at the start and end, to account for Flink's ramp-up and ramp-down.

The queries had different execution durations: the terminal cover dismounted query ran for approximately 60 minutes, the overvoltage query for 90 minutes and the duplicate timestamp query for 120 minutes. Each configuration of the queries were tested five times and the scalability metrics were aggregated, as presented in the figures below.

5.3.1 Terminal cover dismounted query

The following figures depict the measured scalability metrics for the different configurations of the terminal cover dismounted query. One time series graph is included, as they display essentially the same characteristics for each configuration.

Throughput: The throughput distribution demonstrates relatively stable performance across the 10-85% thresholds with a median around 330,000-360,000 tuples/s. A notable outlier, the 99% threshold configuration, shows a similar median, although with a much higher variability.

CPU and memory usage: CPU load distribution reveals consistently high utilization across all configurations, with values predominantly clustering between 98-99%. Notable are the outliers for the 30% threshold configuration down in the 90% range.

The memory usage results demonstrate a similar pattern across all thresholds, with a median between 5400MB and 6500MB.

Latency: Lower configurations (10-30%) maintain relatively low latency (median around 21-24 seconds) with tight distributions, while higher thresholds (50-99%) had elevated latency (median around 46-49 seconds) with increased variance.

5.3.2 Overvoltage query

The following figures depict the measured scalability metrics for the different configurations of the overvoltage query. One time series graph is included, as they display essentially the same characteristics for each configuration.

Throughput: The throughput distribution demonstrates relatively stable performance across the thresholds with a median around 343,000-380,000 tuples/s. All thresholds show a significant amount of outliers in the 600,000-800,000 range.

CPU and memory usage: CPU load distribution reveals consistently high utilization across all configurations with medians between 98-99%. Some outliers are present in all threshold values, the most notable being at 245V with an outlier below 95% CPU load.

The memory usage results demonstrate a similar pattern across all thresholds, with a median between 9200MB and 9400MB.

Latency: The 245-253V thresholds maintain similar median values between 230-270 seconds. Notable here are the extreme outliers, hovering around 1500 seconds for the 245V and 253V configurations, and the 250V configuration having outliers in the 2600 seconds range.

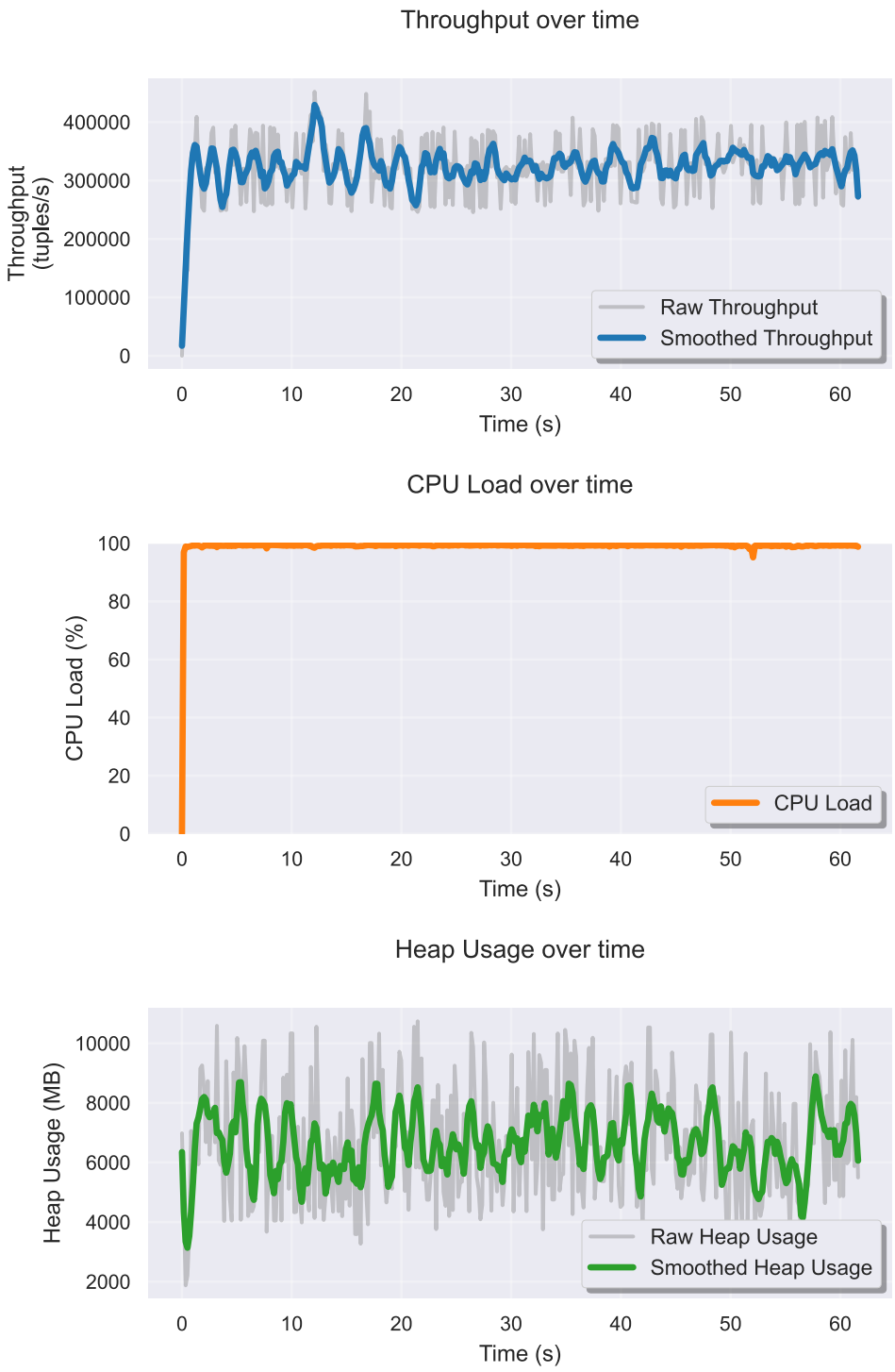

Figure 5.2: Throughput, CPU load, memory over time for the terminal cover dismounted query.

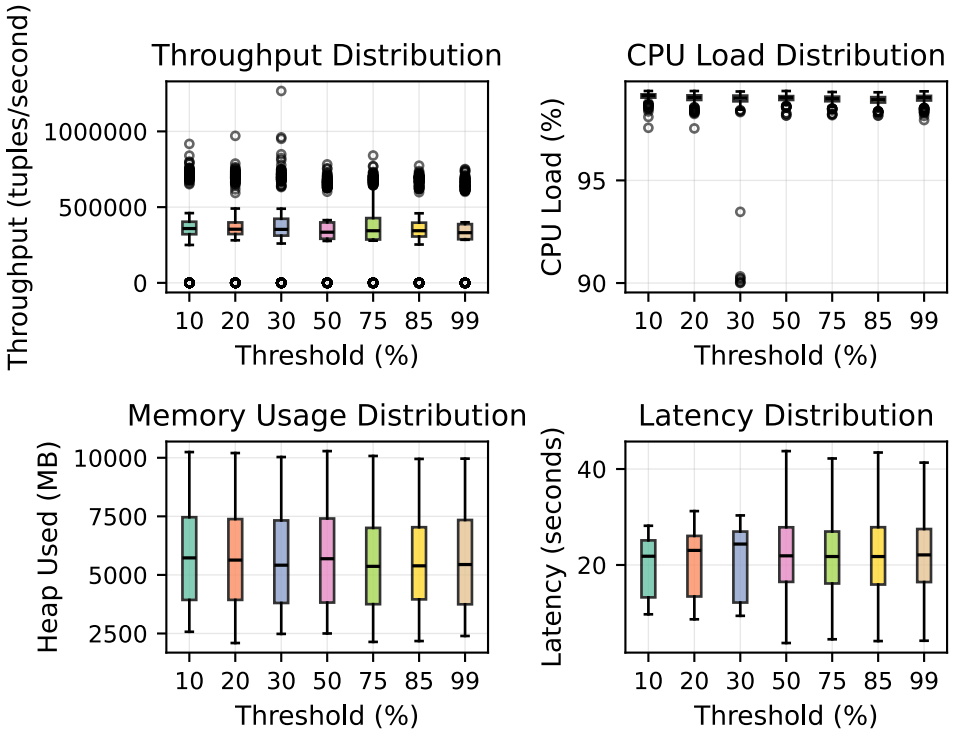

Figure 5.3: Throughput, CPU load, memory, and latency results for the cover queries.

5.3.3 Duplicate timestamp query

The following figures depict the measured scalability metrics for the duplicate timestamp query, in box plot format as well as a timeseries graph.

Throughput: The throughput distribution demonstrates a median around 145,000 tuples/s with a few outliers extending up to the 580,000 range.

CPU and memory usage: CPU load distribution reveals consistently high utilization for this query as well, with a median at 98%. Two outliers between 92-93% are present.

The memory usage shows a median at 7200MB, with one outlier in the 2400MB range.

Latency: The graph shows the median latency to be around 37 seconds with an interquartile range from around 25 to 50 seconds. A few outliers are present with a maximum of 89 seconds.

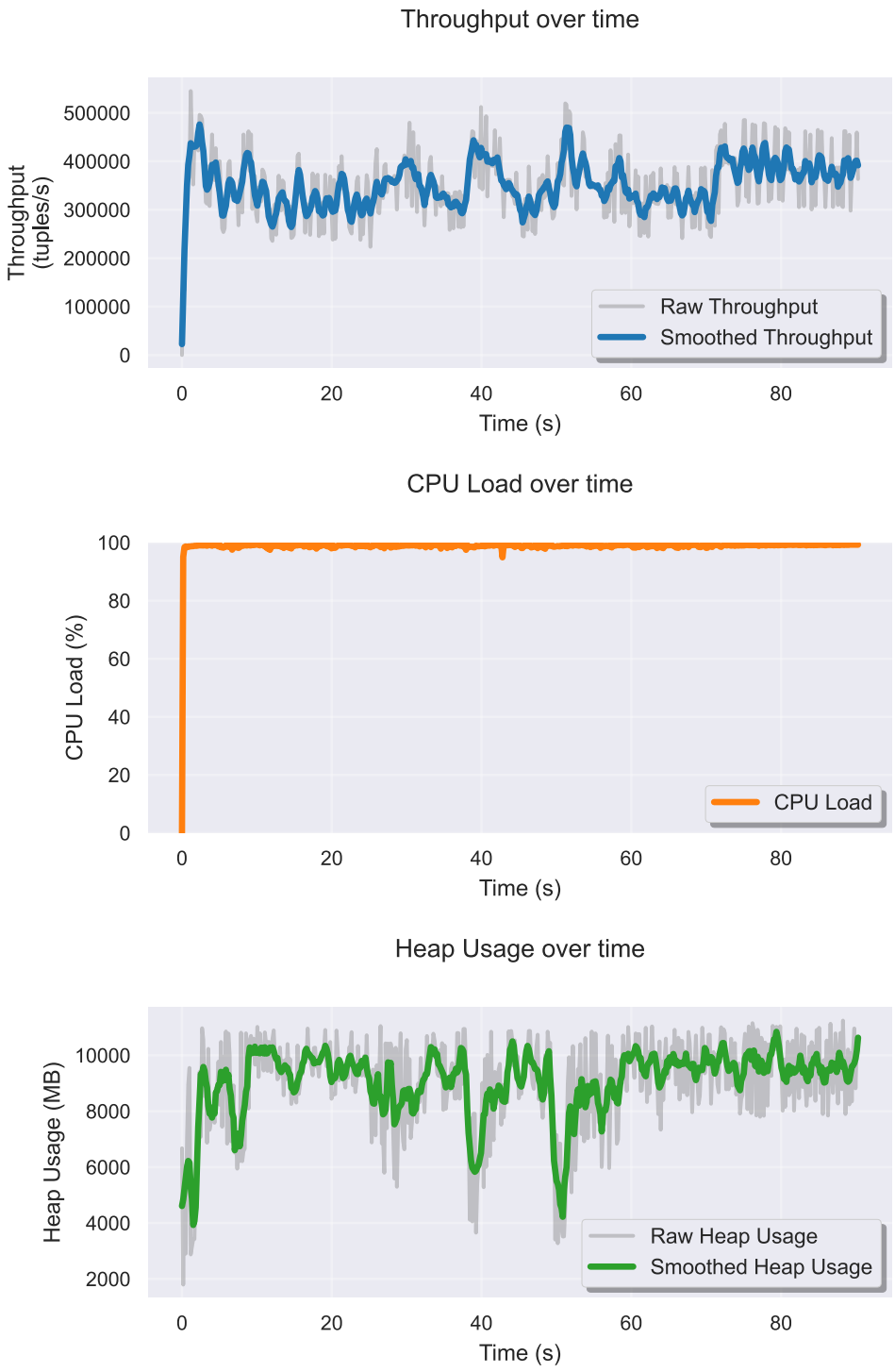

Figure 5.4: Throughput, CPU load, memory over time for the overvoltage query.

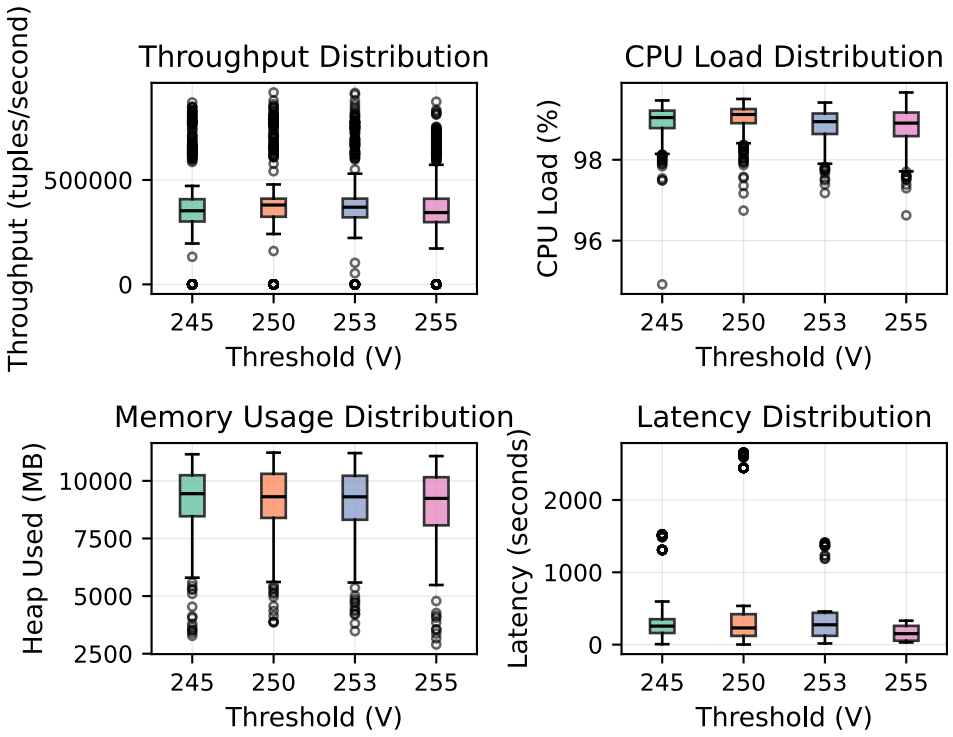

Figure 5.5: Throughput, CPU load, memory, and latency results for the overvoltage queries.

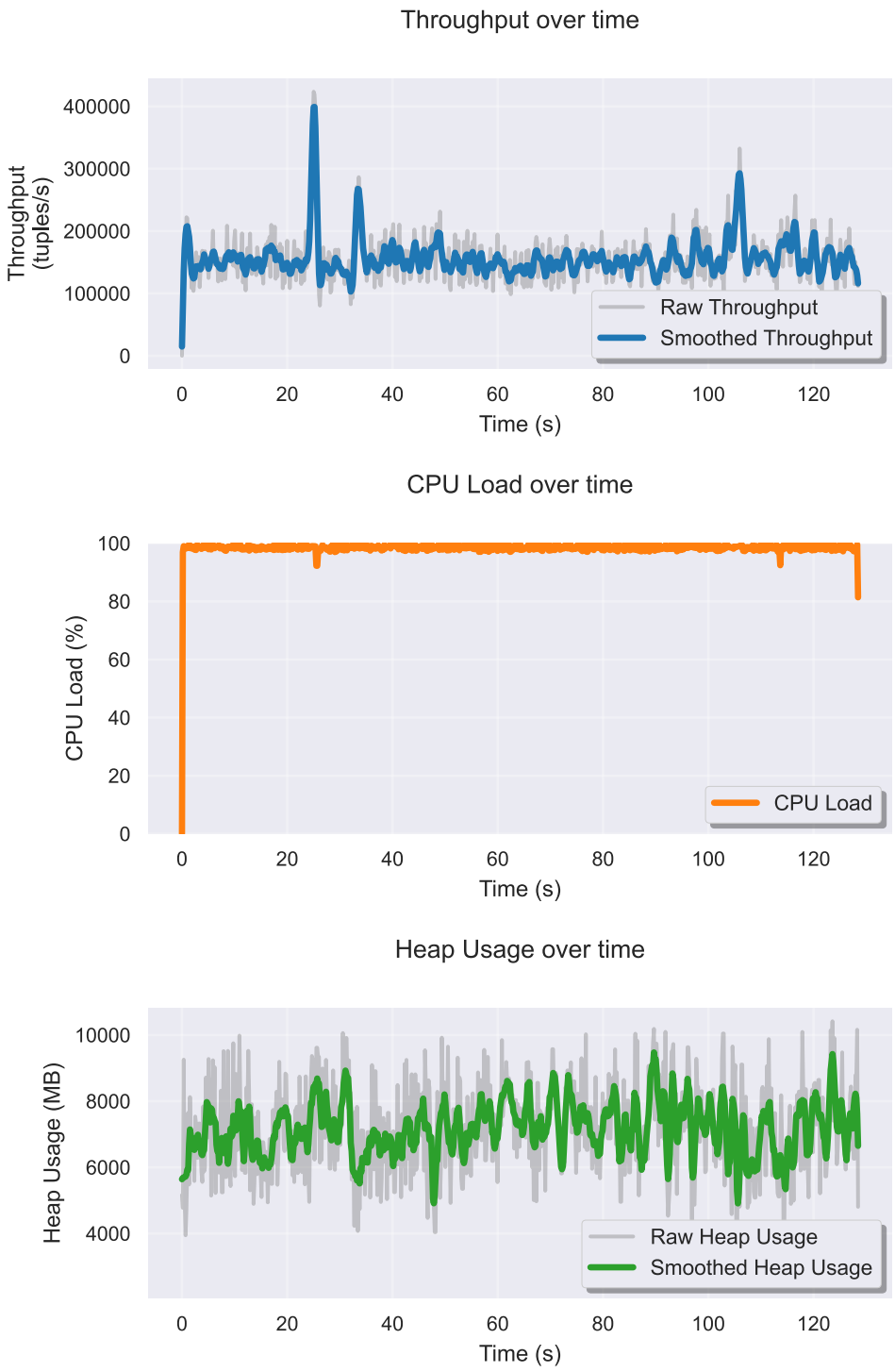

Figure 5.6: Throughput, CPU load, memory over time for the duplicate timestamp query.

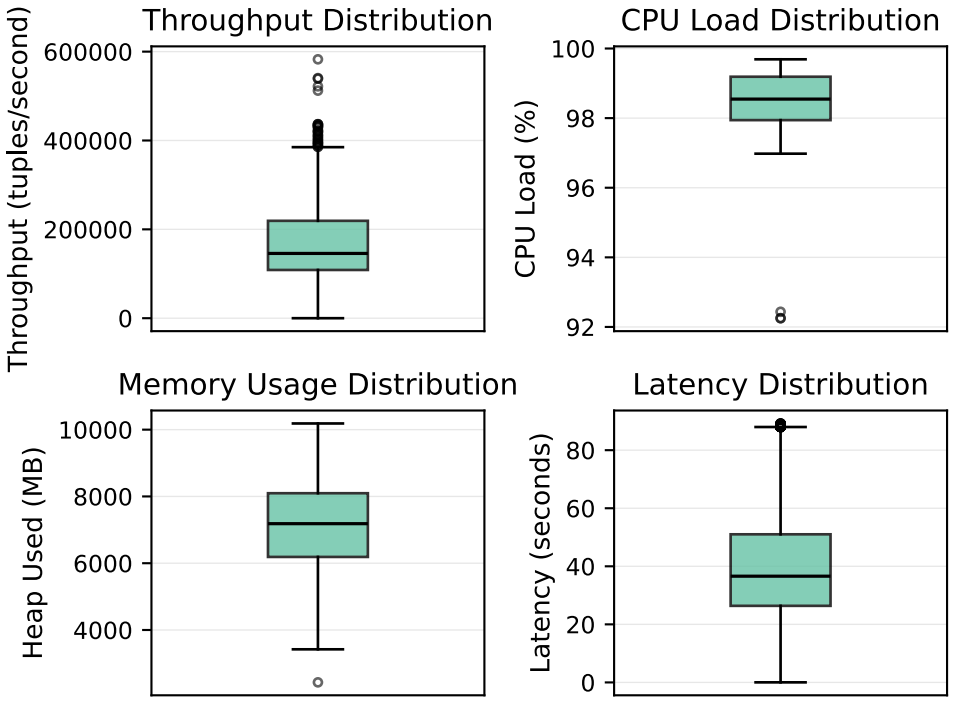

Figure 5.7: Throughput, CPU load, memory, and latency results for the duplicate timestamp query.

6

Discussion

This chapter discusses the results obtained from implementing and evaluating CEP based queries for detecting anomalies in smart meter data at GE.

6.1 Development process

Translating domain expert explanations of meter data patterns into concrete CEP queries presented several challenges. The process required collaboration with analysts at GE to understand the nuances of a meaningful anomaly compared to routine fluctuations. For example, an overvoltage event is not interesting by itself as there could be numerous reasons for it.

The iterative development process revealed the importance of domain knowledge in defining appropriate thresholds, time windows and which specific measurements to use. The overvoltage query required balancing sensitivity to persistent voltage problems against tolerance for brief fluctuations that are operationally insignificant. Similarly, the duplicate timestamp query emerged during the research process when patterns in the data suggested previously unidentified systematic issues. With the knowledge that this should in fact not be present in the data, it was further explored.

6.2 Ratio and precision

The terminal cover dismounted query achieves very aggressive filtering with ratios between 0.46% and 1.00%, meaning that out of 1,413 terminal cover dismounted events, only 6-13 anomalies are generated depending on the threshold configuration. Based on the dataset's date spans, this translates to an estimated 36 daily terminal cover dismounted events being filtered down to less than one anomaly per day. While this extreme reduction minimizes analyst workload, it may be overly aggressive if analysts have capacity to investigate multiple anomalies daily. However, for weekly review this ratio could be optimal, especially since the precision of 46-50% is likely not good enough to treat the anomalies as alerts that require immediate attention.

In contrast, the overvoltage events (the most common meter event) likely require the most aggressive filtering to keep it manageable for an analyst. The analyst can easily prioritize anomalies, starting with the few that are detected with the highest threshold of 255V, as they are also likely to be the ones needing the most immediate

attention. The high precision for this query was expected as the query looks for sustained voltages over the threshold value. By definition an anomalous pattern that is worth looking into according to analysts at GE.

While moderate, the precision results present a significant improvement over the current situation where GE performs no automated filtering. Since analysts currently cannot systematically review all events due to volume constraints, any system that reduces investigation burden while maintaining reasonable precision provides operational value. The challenge lies in finding a balance between filtering aggressiveness and precision for each query type.

6.3 Latency

The observed latency characteristics reveal both the potential and limitations of the current implementation.

For the overvoltage query, specifically with 250V threshold, an investigation with the dataset filtered on the relevant series confirmed that the high volume of concurrent series is not the primary culprit for latency issues. Instead, the problem appears to be that some tuples are read very early in the processing timeline and subsequently become the last contributing input tuple that produces an anomaly, creating significant delays between the earliest and latest contributing events.

Attempts to mitigate this latency through the use of a window, to drop late tuples, before the CEP pattern showed no significant improvement, suggesting that the outliers are not caused by late events. Further investigation is necessary to determine the cause of these outliers.

For the types of anomalies being detected, response times in the minutes to hours range are acceptable, as these issues typically require investigation rather than immediate intervention. In other usages of stream processing, this latency would be unacceptable as one of the reasons for implementing such a system is being alerted immediately when issues are detected in real-time systems.

6.4 Throughput

The observed throughput values provide confidence in the system's scalability for production deployment. The terminal cover dismounted query and overvoltage query shows a sustained throughput of 400,000 tuples per second, with about half for the duplicate timestamp query. With a simple calculation based on the dataset S2, which accurately emulates the true volume of data, an average input rate can be found. With a total of 1,916,413,550 tuples over 6 days, an estimate gives about 4000 tuples per second. In practice there would not be a constant flow but rather larger batches.

However, this processing capacity margin, 400,000 versus 4000, indicates that the system can comfortably handle current data volumes and accommodate growth in meter

deployment or data granularity without requiring additional compute resources.

This throughput capacity also suggests that the system architecture could support additional query types or more complex pattern detection without performance degradation. The scalability characteristics indicate that resource constraints are unlikely to be a limiting factor for practical deployment.

6.5 CPU and memory usage

The system exhibits consistently high CPU utilization throughout processing runs, indicating efficient resource utilization without idle periods. Memory usage peaks at approximately 10GB, which is conservative considering the resources typically available to large utility companies. The stable memory footprint suggests that the system's memory requirements scale predictably with data volume and complexity.

The efficient resource utilization profile supports the feasibility of deploying such systems in production environments without requiring specialized hardware infrastructure beyond what is typically available to utility companies.

6.6 Improvements to the queries

Several opportunities exist to enhance the implemented queries accuracy. The terminal cover dismounted query's consumption averaging mechanism allows for an obvious area of improvement. The current implementation uses simple hourly averages keyed by meter ID and hour of day, but more sophisticated approaches could improve the anomaly detection metrics.

A rolling average, instead of global, that considers logical historical periods (such as seasons) could provide improved consumption estimates. Additionally, including factors such as weekday versus weekend patterns and weather conditions could create more accurate consumption models. Such improvements could reduce the number of false positives while keeping the precision high.

For the duplicate timestamp query, additional mechanisms could help distinguish between different types of timestamp duplication issues, enabling more targeted responses to specific problem categories.

6.7 Implications of duplicate timestamps

The duplicate timestamp query reveals a simple pattern that potentially masks complex underlying issues. The ability to detect these anomalies in near real-time would enable operators to correlate timestamp duplication with other system events, such as collection server reboots, network disruptions, or meter firmware updates. This correlation capability could provide valuable diagnostic information for infrastructure management.

Further improvements could categorize the types of duplicate timestamp anomalies as initial analysis revealed that other parameters in the data displayed different characteristics. This could potentially mean that there are different core issues causing the anomalies. Categorizing could provide more information leading to finding root causes as well as understanding the severity of it. Clearly, the anomalies seen on the top two dates, 2025-02-22 and 2025-03-30, warrants immediate attention as the severity is significantly higher compared to other dates.

Investigations into the anomalies on the 30th of March revealed that they are related to daylight savings time, as many duplications were observed around the specific hour (03:00) where the change of time occurs. The smart meters deployed by GE are not supposed to switch to summer time, so this issue could be caused by the collection servers according to an expert at GE. A similar issue was found in the New York electricity grid by Quilumba-Gudiño [26], where he attributes this to the daylight saving time change.

6.8 Implications of out-of-orderness

The 10 hour out-of-orderness threshold raises concerns about processing latency that would likely be unacceptable in a production system. In a system utilizing a real-time data pipeline, this configuration would introduce a minimum latency of 10 hours before any anomaly could be detected, effectively negating the benefits of stream processing. While the configuration was necessary to accurately evaluate the queries, it showcases a limitation of the dataset that was available at GE.

In truly real-time systems, lower out-of-orderness would reduce state management requirements, improve performance, and enable timely output generation. The current high latency is bounded by the out-of-orderness configuration necessary for the available dataset, but this would be significantly reduced in a streaming application processing live data.

To employ this type of system in a production environment with an acceptable latency it would require the temporal disorder to be addressed at the source. This concern was raised with an expert at GE, who indicated that there is a long-term plan to improve the data collection infrastructure, moving this type of application closer to the source. Ultimately, the goal is to enable real-time applications with minimal latency and a 'process and discard' approach, eliminating the need to store the large volumes of data generated by smart meters.

7

Related Work

This chapter examines existing research related to this thesis.

7.1 Stream processing and CEP in the AMI

The digitalization of the power grid and the widespread deployment of smart meters have generated increasing interest in the use of stream processing and CEP to handle real-time data from the AMI. A variety of research efforts have explored the application of stream processing and CEP in this context, each addressing different aspects such as performance, explainability, privacy, or data validation, often using household energy consumption as a representative use case.

GeneaLog [27] introduced provenance-aware streaming systems capable of capturing fine-grained causal relationships in real time. Ananke [28] extended this direction by enabling live forward provenance while Erebus [29] explored result explainability, both using household energy data. The performance overhead of provenance with data from the Gothenburg grid has been explored in [5].

[30] proposed data structures to improve the concurrency and scalability of windowed aggregation in stream processing, which were tested on AMI data. [31] demonstrated how stream processing can enable scalable, low-latency validation of faulty AMI data. [4] examined how stream processing can be integrated into operational AMI systems, highlighting challenges like handling out-of-order data, configuring watermark strategies, and deploying processing closer to the data source.

[2] presented one of the earliest examples of applying CEP to smart grid monitoring, using a lambda architecture to combine real-time and batch analytics for demand response, voltage anomaly detection, and device coordination. [32] investigated how CEP applications could be automatically synthesized using evolutionary computing. Although their experiments focused on different data, they note that the approach is applicable to systems like AMI, particularly where system experts possess deep domain knowledge but lack programming skills.

While prior work establishes the relevance of stream processing and CEP in AMI systems, it often focuses on isolated data types, assumes ordered input, or uses synthetic or lab-scale datasets. In contrast, this thesis addresses the challenges and opportunities introduced by upgraded smart meters, which now produce both high-frequency time series data and real-time events. By working with unaltered,

production-scale data, this work shows how state-of-the-art tools like Apache Flink and FlinkCEP can be used to correlate and filter these streams under realistic conditions. This approach focuses on the operational feasibility of CEP for anomaly detection by exploring the capabilities of the new meters but also the challenges of the current system with large amounts of false alarms and out-of-order data.

7.2 Anomaly detection in AMI data

Anomaly detection in AMI data has previously been studied, particularly in the context of identifying faulty meters, energy theft, or installation errors. Multiple techniques have been proposed, ranging from rule-based analysis to advanced machine learning.

A previous research contribution at GE was one of the first works to use voltage data to identify broken smart meters [3]. This thesis expands on that work by integrating voltage data with meter events to identify potentially broken meters. However, this work takes a different perspective, as the aim is to filter out events generated by the meters themselves since the meters now have the capability to send alerts about high voltage levels.

Similarly, a previous master's thesis at GE has explored the detection of change-points to identify faulty meter installations or energy theft [5]. This thesis uses a similar type of query, looking at the average consumption before and after an event. The difference being that this thesis incorporates events from the meters themselves during operation, rather than using information about when meters are changed from the utility company.

Previously, utilities relied solely on physical seals to detect unauthorized access to meters. This method was not reliable or automated, as discussed in [33], which also notes that several theft-related attacks require physical tampering. The new meter events provide an improvement in the detection of tampering, but they do not differentiate between routine maintenance and unauthorized access. This thesis builds on this new capability, combining it with electricity consumption data to enable a mechanism to classify the events.

Additionally, [34] developed methods for electricity theft detection using customer consumption patterns, applying machine learning to AMI data. However, their approach relied on batch processing of historical data rather than stream processing and CEP. More recent work has explored deep learning approaches to anomaly detection. [35] proposed wide and deep convolutional neural networks for electricity theft detection, while [36] investigated artificial neural networks for smart grid energy fraud detection. A natural next step is to combine the capabilities of CEP with machine learning capabilities as proposed in a framework by [37]. The evolutionary computing approach in [32] could be used to assist domain experts in automatically generating CEP rules without requiring programming expertise, which can be used to detect anomalies. Finally, [38] introduced a two-tier intrusion detection framework for AMI systems, using probabilistic models to detect stealthy attacks such as energy data exfiltration.

Previous work on anomaly detection in AMI data has either laid the groundwork for what detection capabilities smart meters should support, or has been constrained by meters that provided limited data compared to the new ones. With the introduction of meters that now report certain anomalies directly, new opportunities for real-time detection have emerged. However, this introduces new challenges as the raw alerts alone proved to be insufficient for actionable detection due to their ambiguity and volume. This thesis addresses that gap by combining real-time meter events with time series data to detect operational anomalies more effectively, using CEP to refine, contextualize, and reduce false positives. This work serves as a case study, exploring how the upgraded infrastructure can be leveraged in practice.

7.3 Data quality issues

While not part of the initial aim of the thesis, data quality issues in the available dataset proved to be a challenge necessary to tackle. Similar data issues have been explored previously. Wang et al. [39] conducted a review of smart meter data analytics, highlighting bad data as an issue and the impact it has on analytical accuracy. Similarly, [40] identifies three types of data quality issues in a smart grid environment: noise data, incomplete data, and outlier data. Finally, [31] also highlights these issues and proposes validation analysis to combat them.

This thesis contributes by discovering and documenting a systematic issue with duplicate timestamps, which was not previously known at GE. Notably, similar issues have been discussed by [26], who also identified daylight saving time changes as a contributing factor to duplicate data. These findings demonstrate the value of CEP, not only for detecting known anomalies but also for uncovering new types of data quality issues that may indicate underlying infrastructure problems.

8

Conclusion

This thesis investigated how AMI data, such as smart meter events and time-series measurements, can be validated and analyzed using CEP to detect meaningful anomalies. Three distinct use cases were implemented: the terminal cover dismounted query, the overvoltage query and the duplicate timestamp query.

The terminal cover dismounted query shows how CEP can correlate meter events with subsequent measurement changes, using interval joins to link physical tampering events with consumption anomalies occurring within a five-hour window. This approach achieved filtering ratios between 0.46% and 1.00%, reducing 1,413 terminal cover events to 6-13 actionable anomalies.

The overvoltage query demonstrates CEP's ability to detect sustained abnormal conditions rather than momentary spikes, requiring eight consecutive voltage readings above threshold across a two-hour period. This approach achieved filtering ratios from 0.006% to 1.44%, depending on voltage thresholds, reducing 98,547 overvoltage events to manageable numbers for analyst review.

The duplicate timestamp query illustrates CEP's capability to detect systematic data collection issues through simple sequence pattern matching, identifying nearly 269,000 affected meters on peak dates and revealing previously unknown problems.

The evaluation of these queries shows strong scalability characteristics that support practical deployment of such a system. The system demonstrates excellent throughput capacity, processing approximately 400,000 tuples per second which is significantly higher than the estimated actual data flow of 4,000 tuples per second. This processing margin provides room for growth in meter deployment or data granularity. CPU utilization remains consistently high during processing, indicating resource usage without idle periods. Memory usage peaks at approximately 10GB, which is conservative for enterprise utility infrastructure. While most of the reported anomalies exhibit latencies that are acceptable for operators at GE, there are outliers with unexpectedly high latency. It may be related to the characteristics of the dataset, such as out-of-orderness, but further analysis is required to confirm this. This would represent a data quality limitation rather than a fundamental system constraint and is likely to be improved if this type of CEP system is deployed closer to the source to remain unaffected by issues present in the layers of the data collection systems.

These findings confirm that CEP can be a powerful tool for analyzing AMI data, notifying operators about new instances of known issues as well as supporting the

discovery of previously unknown problems. The results demonstrate the potential of CEP systems to enhance detection of important anomalies related to smart meters, while also allowing analyst to prioritize and streamline the manual work of investigation. Future work could further explore this approach, developing more advanced systems and moving toward real-time deployments. This could enable timely, high-resolution insights, leading to a more proactive and efficient grid operation.

8.1 Future work

Several areas warrant further investigation to enhance the practical applicability of CEP-based anomaly detection for smart grid monitoring. Code quality improvements should be prioritized before production deployment, including better error handling, monitoring capabilities, and configuration management. The prototypes developed in this work should be seen as rough drafts used to explore the possibilities of using FlinkCEP at GE. As such, production-ready applications likely need to be created from the ground up by more experienced developers.

Before implementing real-time processing, consideration should be given to storing processed data in efficient formats such as Parquet for longer-term analysis. This approach was found to be valuable during development for testing assumptions and discovering new patterns like the duplicate timestamp anomaly. Furthermore, describing queries with an SQL-like syntax is likely easier to start with as it is more accessible to non-developers, bridging the communication gap between expert analysts and developers. Insights from this could then be used to deploy a streaming application. A hybrid approach that combines real-time anomaly detection with batch analysis capabilities could provide both immediate alerting and deeper analytical insights.

Categorizing the specific issues identified by the duplicate timestamp query could provide valuable insights into infrastructure problems and guide maintenance prioritization. This analysis could help distinguish between meter hardware issues, communication problems, and data collection system errors.

Continued exploration of the data surrounding generated anomalies could reveal additional patterns suitable for automated detection. The current queries represent an initial exploration of the possibilities, but the rich dataset likely contains other valuable signals that could be incorporated into a more comprehensive anomaly detection system. Moreover, there are several more meter event types not discussed in this work, some of these may hold the key to discovering other anomalies.

Finally, integration with external data sources, such as weather information [21], [41] and maintenance schedules [5] could enhance the accuracy or context of anomaly detection. Machine learning and evolutionary computing [32] approaches could be explored to automatically learn consumption patterns and voltage characteristics, potentially identifying more subtle anomalies than rule-based approaches.

Bibliography

[1] J. S. Guerrero-Prado, W. Alfonso-Morales, E. Caicedo-Bravo, B. Zayas-Pérez, and A. Espinosa-Reza, "The Power of Big Data and Data Analytics for AMI Data: A Case Study," *Sensors*, vol. 20, no. 11, 2020, ISSN: 1424-8220. DOI: 10.3390/s20113289. [Online]. Available: https://www.mdpi.com/1424-8220/20/11/3289.

[2] G. Liu, W. Zhu, C. Saunders, F. Gao, and Y. Yu, "Real-time Complex Event Processing and Analytics for Smart Grid," *Procedia Computer Science*, vol. 61, pp. 113–119, 2015, Complex Adaptive Systems San Jose, CA November 2-4, 2015, ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2015.09.169. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050915029993.

[3] J. van Rooij, V. Gulisano, and M. Papatriantafilou, "LoCoVolt: Distributed Detection of Broken Meters in Smart Grids through Stream Processing," in *Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems*, ser. DEBS '18, ACM, Jun. 2018, pp. 171–182. DOI: 10.1145/3210284.3210298. [Online]. Available: http://dx.doi.org/10.1145/3210284.3210298.

[4] Joris van Rooij, "Data stream processing meets the Advanced Metering Infrastructure: possibilities, challenges and applications," Licentiate Thesis, Chalmers University of Technology, 2020.

[5] J. Taube and W. Johnsson, "Streaming Analytics with Provenance in the Advanced Metering Infrastructure," Department of Computer Science and Engineering, Master's thesis, Chalmers University of Technology, Gothenburg, Sweden, 2022. [Online]. Available: https://odr.chalmers.se/handle/20.500.12380/305852.

[6] Infrastrukturdepartementet, *Förordning (1999:716) om mätning, beräkning och rapportering av överförd el*, https://www.riksdagen.se/sv/dokument-och-lagar/dokument/svensk-forfattningssamling/forordning-1999716-om-matning-berakning-och_sfs-1999-716/, [Accessed 28-02-2025].

[7] Kamstrup, *Kamstrup OMNIA® e-meter three-phase | Data Sheet, p.6*, https://documentation.kamstrup.com/docs/OMNIA_e_meter_three_phase/en-GB/Data_sheet/CONT52414B8F459B455FABDD08B34E8647A6/?page=6, [Accessed 03-03-2025], 2024.

[8] I. Botan, R. Derakhshan, N. Dindar, L. Haas, R. J. Miller, and N. Tatbul, "SECRET: a model for analysis of the execution semantics of stream processing systems," *Proc. VLDB Endow.*, vol. 3, no. 12, pp. 232–243, Sep. 2010, ISSN:

2150-8097. DOI: 10.14778/1920841.1920874. [Online]. Available: https://doi.org/10.14778/1920841.1920874.

[9] T. Akidau, R. Bradshaw, C. Chambers, *et al.*, "The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing," *Proceedings of the VLDB Endowment*, vol. 8, pp. 1792–1803, 2015.

[10] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache Flink: Stream and Batch Processing in a Single Engine," *IEEE Data Engineering Bulletin*, vol. 38, Jan. 2015.

[11] Apache Software Foundation, *DataStream API: Joining*, https://nightlies.apache.org/flink/flink-docs-release-1.20/docs/dev/datastream/operators/windows/, [Accessed 21-03-2025], 2025.

[12] Apache Software Foundation, *Apache License*, https://www.apache.org/licenses/LICENSE-2.0, [Accessed 22-05-2025].

[13] V. Gulisano, D. Palyvos-Giannas, B. Havers, and M. Papatriantafilou, "The role of event-time order in data streaming analysis," in *Proceedings of the 14th ACM International Conference on Distributed and Event-Based Systems*, ser. DEBS '20, Montreal, Quebec, Canada: Association for Computing Machinery, 2020, pp. 214–217, ISBN: 9781450380287. DOI: 10.1145/3401025.3404088. [Online]. Available: https://doi.org/10.1145/3401025.3404088.

[14] J. van Rooij, V. Gulisano, and M. Papatriantafilou, "TinTiN: Travelling in time (if necessary) to deal with out-of-order data in streaming aggregation," in *Proceedings of the 14th ACM International Conference on Distributed and Event-Based Systems*, ser. DEBS '20, Montreal, Quebec, Canada: Association for Computing Machinery, 2020, pp. 141–152, ISBN: 9781450380287. DOI: 10.1145/3401025.3401769. [Online]. Available: https://doi.org/10.1145/3401025.3401769.

[15] Apache Software Foundation, *Fixed Amount of Lateness*, https://nightlies.apache.org/flink/flink-docs-release-1.20/docs/dev/datastream/event-time/built_in/, [Accessed 31-03-2025].

[16] Apache Software Foundation, *Allowed Lateness*, https://nightlies.apache.org/flink/flink-docs-master/docs/dev/datastream/operators/windows/, [Accessed 31-03-2025].

[17] Apache Software Foundation, *DataStream API: Joining*, https://nightlies.apache.org/flink/flink-docs-release-1.20/docs/dev/datastream/operators/joining/, [Accessed 15-03-2025], 2025.

[18] S. S. Kumar and S. Agarwal, "Rule based complex event processing for IoT applications: Review, classification and challenges," *Expert Systems*, vol. 41, no. 9, e13597, 2024. DOI: https://doi.org/10.1111/exsy.13597. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/exsy.13597. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/exsy.13597.

[19] V. Gulisano and A. Margara, "Aggregates are all you need (to bridge stream processing and Complex Event Recognition)," in *Proceedings of the 18th ACM International Conference on Distributed and Event-Based Systems*, ser. DEBS '24, Villeurbanne, France: Association for Computing Machinery, 2024, pp. 66–

77, ISBN: 9798400704437. DOI: 10.1145/3629104.3666032. [Online]. Available: https://doi.org/10.1145/3629104.3666032.

[20] Apache Software Foundation, *FlinkCEP - Complex event processing for Flink*, https://nightlies.apache.org/flink/flink-docs-release-1.20/docs/libs/cep/, [Accessed 21-05-2025], 2025.

[21] U. E. I. A. (EIA), *Hourly electricity consumption varies throughout the day and across seasons*, https://www.eia.gov/todayinenergy/detail.php?id=42915, [Accessed 22-05-2025], 2020.

[22] Microsoft, *Basv2 size series - Azure Virtual Machines*, https://learn.microsoft.com/en-us/azure/virtual-machines/sizes/general-purpose/basv2-series, [Accessed 21-05-2025].

[23] Apache Software Foundation, *Deployment Config*, https://nightlies.apache.org/flink/flink-docs-release-1.20/docs/deployment/config/, [Accessed 21-05-2025], 2025.

[24] Apache Software Foundation, *Parallel Execution*, https://nightlies.apache.org/flink/flink-docs-release-1.20/docs/dev/datastream/execution/parallel/, [Accessed 21-05-2025], 2025.

[25] Apache Software Foundation, *Metrics*, https://nightlies.apache.org/flink/flink-docs-release-1.20/docs/ops/metrics/, [Accessed 21-05-2025], 2025.

[26] F. L. Quilumba-Gudiño, "Using Advanced Metering Infrastructure Data for Smart Grid Development," PhD Dissertation, The University of Texas at Arlington, Arlington, TX, May 2014.

[27] D. Palyvos-Giannas, V. Gulisano, and M. Papatriantafilou, "Genealog: Fine-grained data streaming provenance in cyber-physical systems," *Parallel Computing*, vol. 89, p. 102 552, 2019, ISSN: 0167-8191. DOI: https://doi.org/10.1016/j.parco.2019.102552. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167819119301437.

[28] D. Palyvos-Giannas, B. Havers, M. Papatriantafilou, and V. Gulisano, "Ananke: A streaming framework for live forward provenance," *Proc. VLDB Endow.*, vol. 14, no. 3, pp. 391–403, Nov. 2020, ISSN: 2150-8097. DOI: 10.14778/3430915.3430928. [Online]. Available: https://doi.org/10.14778/3430915.3430928.

[29] D. Palyvos-Giannas, K. Tzompanaki, M. Papatriantafilou, and V. Gulisano, "Erebus: Explaining the outputs of data streaming queries," *Proc. VLDB Endow.*, vol. 16, no. 2, pp. 230–242, Oct. 2022, ISSN: 2150-8097. DOI: 10.14778/3565816.3565825. [Online]. Available: https://doi.org/10.14778/3565816.3565825.

[30] V. Gulisano, Y. Nikolakopoulos, D. Cederman, M. Papatriantafilou, and P. Tsigas, "Efficient data streaming multiway aggregation through concurrent algorithmic designs and new abstract data types," *ACM Trans. Parallel Comput.*, vol. 4, no. 2, Oct. 2017, ISSN: 2329-4949. DOI: 10.1145/3131272. [Online]. Available: https://doi.org/10.1145/3131272.

[31] V. Gulisano, M. Almgren, and M. Papatriantafilou, "Online and scalable data validation in advanced metering infrastructures," *IEEE PES Innovative Smart*

Grid Technologies Conference Europe, vol. 2015, Jan. 2015. DOI: 10.1109/ISGTEurope.2014.7028740.

[32] G. Appetito, E. Medvet, and V. Gulisano, "Automated discovery of cep applications with evolutionary computing," in *Proceedings of the 19th ACM International Conference on Distributed and Event-Based Systems*, ser. DEBS '25, Association for Computing Machinery, 2025, pp. 33–38, ISBN: 9798400713323. DOI: 10.1145/3701717.3730548. [Online]. Available: https://doi.org/10.1145/3701717.3730548.

[33] S. McLaughlin, D. Podkuiko, and P. McDaniel, "Energy theft in the advanced metering infrastructure," in *Critical Information Infrastructures Security*, E. Rome and R. Bloomfield, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 176–187, ISBN: 978-3-642-14379-3.

[34] P. Jokar, N. Arianpoo, and V. C. M. Leung, "Electricity Theft Detection in AMI Using Customers' Consumption Patterns," *IEEE Transactions on Smart Grid*, vol. 7, no. 1, pp. 216–226, 2016. DOI: 10.1109/TSG.2015.2425222.

[35] Z. Zheng, Y. Yang, X. Niu, H.-N. Dai, and Y. Zhou, "Wide and Deep Convolutional Neural Networks for Electricity-Theft Detection to Secure Smart Grids," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 4, pp. 1606–1615, 2018. DOI: 10.1109/TII.2017.2785963.

[36] V. Ford, A. Siraj, and W. Eberle, "Smart grid energy fraud detection using artificial neural networks," in *2014 IEEE Symposium on Computational Intelligence Applications in Smart Grid (CIASG)*, 2014, pp. 1–6. DOI: 10.1109/CIASG.2014.7011557.

[37] M. U. Simsek, F. Y. Okay, and S. Ozdemir, "A deep learning-based CEP rule extraction framework for IoT data," *The Journal of Supercomputing*, vol. 77, no. 8, pp. 8563–8592, 2021, ISSN: 1573-0484. DOI: 10.1007/s11227-020-03603-5. [Online]. Available: https://doi.org/10.1007/s11227-020-03603-5.

[38] V. Gulisano, M. Almgren, and M. Papatriantafilou, "Metis: A two-tier intrusion detection system for advanced metering infrastructures," in *International Conference on Security and Privacy in Communication Networks*, J. Tian, J. Jing, and M. Srivatsa, Eds., Cham: Springer International Publishing, 2015, pp. 51–68, ISBN: 978-3-319-23802-9.

[39] Y. Wang, Q. Chen, T. Hong, and C. Kang, "Review of Smart Meter Data Analytics: Applications, Methodologies, and Challenges," *IEEE Transactions on Smart Grid*, vol. 10, no. 3, pp. 3125–3148, 2019. DOI: 10.1109/TSG.2018.2818167.

[40] W. Chen, K. Zhou, S. Yang, and C. Wu, "Data quality of electricity consumption data in a smart grid environment," *Renewable and Sustainable Energy Reviews*, vol. 75, pp. 98–105, 2017, ISSN: 1364-0321. DOI: https://doi.org/10.1016/j.rser.2016.10.054. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1364032116307109.

[41] J. Kang and D. M. Reiner, "What is the effect of weather on household electricity consumption? empirical evidence from ireland," *Energy Economics*, vol. 111, p. 106 023, 2022, ISSN: 0140-9883. DOI: https://doi.org/10.1016/

j.eneco.2022.106023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S014098832200189X.