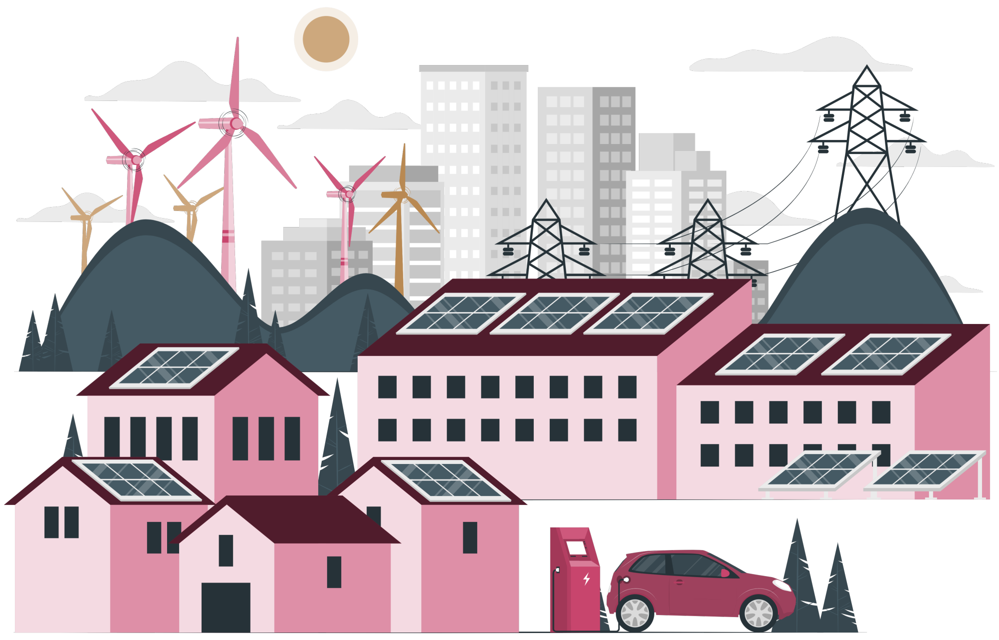

Data streaming provenance in advanced metering infrastructures

In collaboration with Göteborg Energi

Master's thesis in Computer science and engineering

Zozk Mohamed

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

Data streaming provenance in advanced metering infrastructures

In collaboration with Göteborg Energi

Zozk Mohamed

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Data streaming provenance in advanced metering infrastructures
In collaboration with Göteborg Energi
Zozk Mohamed

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

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2023

Data streaming provenance in advanced metering infrastructures
In collaboration with Göteborg Energi
Zozk Mohamed
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Increasing volumes of data in digital systems have made the traditional approach of gathering and storing all the data while analyzing it in bulks at periodic intervals challenging and costly. One such field is the electric grid market, which has started modernizing its aging grids into smart grids where Advanced Metering Infrastructures (AMIs) play a vital role. Within AMIs, old meters are replaced with smart meters that are able to collect data more often and measure more properties than the old meters. However, they also produce a higher volume of data. Stream processing where data is analyzed continuously before being stored, can therefore be of interest as data can be heavily reduced before storage. The downside of this approach is that the traceability of data is lost. A technique that can solve this is called stream provenance which can be used to get the source data that contributed to the output data from a stream processing application. However, stream provenance is an understudied problem that can decrease performance when used. The purpose of this thesis is to study stream provenance by developing a streaming application that makes use of provenance. The application is evaluated by measuring several metrics to determine how performance is affected. The project is conducted at Göteborg Energi (GE), one of Sweden's biggest energy utility companies. The objective is to develop a prototype extension to GE's current stream-processing application that can detect faulty meters and use stream provenance to report them. The development processes and evaluation of the application are covered in this report. The application is developed through a Stream Processing Engine (SPE) called Apache Flink and a stream provenance framework called Ananke. Two versions are created, one with provenance and another without. Performance metrics like CPU utilization, memory consumption, latency, and throughput are measured. The result showed that provenance decreases throughput by 10.4% and increases memory consumption by 8.8%, latency by 10.4%, and CPU utilization by 238.1%. Several reasons behind the result are discussed in the report, along with the implications it can have for an application. Although there is an added overhead with provenance, it can still be beneficial for some types of applications. For example, an application where time is not crucial and good access to resources is possible, like the one developed in this thesis.

Keywords: Advanced Metering Infrastructure, Ananke, Apache Flink, Göteborg Energi, Provenance, Stream processing, Stream Processing Engine.

Acknowledgements

I would like to express my deepest appreciation to my supervisor at Chalmers, Vincenzo Massimiliano Gulisano, for his generously provided expertise and advice. I would also like to express my deepest gratitude to my supervisor at Göteborg Energi, Joris van Rooij, for his invaluable feedback and patience. Additionally, I am also grateful for the welcoming and nice team at Göteborg Energi. Lastly, I would like to thank my family and friends for their support and belief in me.

Zozk Mohamed, Gothenburg, November 2023

Contents

List of Figures

List of Tables

1 Introduction

This chapter introduces some background information about the thesis's relevance and the current problem in the field. The aim is then presented to define the goal of the project and how it will deal with the problem. Lastly, some limitations are presented on what the thesis will not cover.

1.1 Background

Sweden's electrification began in 1893 when electricity was transferred for the first time from a small hydroelectric station to a mine in Dalarna, at a distance of 15 km [1]. Since then, electrification has evolved, and today's society and economy depend heavily on access to both affordable and reliable electric power. For example, to operate infrastructure, run health care, and light our world. Due to several reasons, the climate crisis being one of them, many companies and individuals have chosen to switch out fossil-based fuels and use products that require electricity instead. This has led to an escalation in global electricity demand. For example, Sweden's electric demand will according to Swedenergy's latest report increase by 94 percent until 2045, from 170 TWh in 2022 to 330 TWh in 2045 [2]. Today's power grids are still based on the same design that we had more than 100 years ago and might not support future growth in electric demand as they are optimized for centralized power generation and demand-driven response [3]. However, thanks to the development of information systems and communication technology, many companies have started to modernize their aging power systems into smart grids. They offer far more opportunities than the previous types of grid did. For example, a smart grid offers distributed two-way transmission, reliability, sustainability, consumer involvement, security, and most importantly energy efficiency [4].

One key component that has increased the use of smart grids is Advanced Metering Infrastructure (AMI). AMI has improved the electricity metering system by replacing old mechanical meters with Smart Meters (SMs). They are placed all over the grid and can remotely collect more detailed data on electric usage compared to mechanical meters. In addition, SMs can help energy utility companies monitor their grid, avoid overloading components of the grid, provide a faster diagnosis in case of outages, and accurately bill customers based on their electricity consumption [4]. The current regulation requires only reading every hour, and most meters are limited to this. However, from January 2025, this will change with a new law that puts more requirements on the meters. For example, they will need to register a reading every 15 minutes and measure more properties than the previous regulation required [5]. This has forced many energy utility companies to update their current AMI systems.

Within AMI, a continuous and large volume of data is exchanged between devices, which can consist of millions of messages daily [6]. The traditional and well-known way to process data is batch processing, where all data is moved and stored in a database to later be processed by the application. This method is well-known in the industry and is often the first option. However, it can be very costly and sometimes even impossible to only rely on batch processing in AMI as data can be inserted too fast or be too big. This is something observed in many digital and cyber-physical systems [6] that leads to the need to move the analyzing part close to the source before moving and storing the data. This can reduce the volume of data that is transmitted but also make it possible to react more quickly to critical or unusual situations.

Many utility providers have started leveraging other data analysis paradigms besides the traditional batch processing ones to handle such continuous and large amounts of data. One such paradigm is data streaming which continuously performs queries on unbounded data without the need to store it in databases. These queries are not the same as in databases where they are performed at a point in time on bounded data, instead, they are applied continuously to process information on the fly, updating their computation and quickly producing results accordingly [6]. Once data is processed, it is passed off to either an application, data store or another framework. In some streaming applications, it can be interesting to maintain the associated source data for further analysis by for example a human supervisor. For example when data is transformed and analyzed to find faulty data. One technique that can achieve this is called streaming provenance. It allows output from a stream processing application to be traced back to the source data that contributed to it. This makes it possible for a data analyst to analyze the data and find the cause of the output [7].

1.2 Problem statement

Although some research has been carried out on smart grids and AMI, there have been few empirical investigations into the kinds of applications that can be developed and the tools that can support them. With the increased volume of data that smart grids entail, it will no longer be practical for energy utility companies to only rely on batch processing, as it can be very costly and sometimes even impossible to do. Instead, other paradigms like stream processing that can reduce unnecessarily large volumes of raw data into smaller sets of useful information, need to be looked into. However, compared to batch processing, it is not possible to view all data at once, as data is ingested and processed continuously. This makes it difficult to find data's origin [8]. Writing applications that both reduce data volume and keep important raw data is left to users and is not trivial. Therefore, provenance could be a solution. However, according to several research papers, [7]–[10] provenance has mostly been investigated in the context of databases and is less studied in the context of stream processing. Problems regarding the implementation of provenance and the overhead it has to an application that affects it in terms of performance and memory consumption, need to be looked into.

This masters thesis is in collaboration with one of the biggest energy utility companies in Sweden, namely Göteborg Energi (GE) [11]. They are like many other energy utility companies switching out their meters for new generation SMs and thus will have to face the problem with the increased data. GE currently uses stream processing for an analysis application that makes forecasts of future meter readings in the grid. For each meter, it shows the current load together with a forecast that is based on historical data. However, there are events that the forecasting model used by the application is not designed to handle. For example, faulty installed meters, or ongoing maintenance work. Among several, one consequence of these kinds of events is that the forecasts will have large derivations from the outcome meter reading. In addition, these kinds of events will affect future forecasts by making them less accurate as they will be based on faulty data received from these events.

1.3 Aim

The thesis aims to use GE's current analysis platform to study, implement, design, and evaluate a prototype extension that uses stream processing and provenance to generate alerts for meters that are suspected to be faulty. The faulty meters should be identified by detecting large deviations between the outcome reading and the forecast generated by GE's current application. The large derivation between these two can possibly imply a faulty meter. Already known faulty meters are not of interest and therefore the extension should not generate any trigger for them. The extension should be able to handle the increased data volume from the new meters, according to the new requirements to be enforced starting from 2025 [5]. In addition, it should be possible to activate and deactivate provenance to evaluate how the extension is affected. The evaluation of the extension should measure several metrics to help determine its usefulness in terms of performance. In the end, the thesis aims to help future research in the area by identifying both opportunities and challenges with using stream processing and provenance in AMI.

1.4 Constraints

For this thesis, meters are considered to be faulty when readings from the meter deviate significantly from the forecast as this can be due to a faulty installed meter, illegal modification, or a malfunction. However, there can be other reasons for these derivations, for example, a bad forecast. In addition, there can be other ways for a meter to be considered faulty. For example, negative consumption. These are left out in this thesis.

No method for streaming provenance is developed or implemented in this masters thesis due to time constraints. Instead, already developed frameworks such as GeneaLog [7] and its successor Ananke [8] are used as they support the current system that GE uses. Another limitation is that this thesis does not provide a comprehensive review of other non-streaming applications and frameworks that the extension may interact with. Although AMI is quite broad and may include other utilities such as gas, water, or district heating, this thesis only focuses on data related to the electrical grid.

1.5 Thesis outline

The rest of the report is organized as follows:

Chapter 2, Technical Background presents important theoretical background required to understand the rest of the report. This includes a review of AMI, streaming provenance, and important concepts within stream processing.

Chapter 3 Problem Definition describes GE's current streaming application that generates forecasts. The problem with this system is explained and the requirements the new prototype extension should fulfill in order to solve it. Several questions are also presented that the thesis aims to answer. Lastly, the performance metric that the evaluation aims to measure is defined.

Chapter 4 Methodology describes the tools used in the project, how the application works, and how the evaluation is conducted.

Chapter 5 Results describes the result from the evaluation by presenting box plots, histograms, and tables over the obtained result and discussing their meaning.

Chapter 6 Discussion contains reflections on the development process, the challenges faced, the limitations of the project, and how it can be further developed.

Chapter 7 Conclusion concludes the thesis by summarizing the key findings and their impact on future research.

2

Technical Background

This chapter describes important elements used in the rest of the thesis. Firstly the organizing of Advanced Metering Infrastructure is described as well as its pros and cons. The next section goes through relevant concepts in stream processing and the last section is about stream provenance and how it works.

2.1 Advanced Metering Infrastructure

As the introduction mentions, one key component of the increased use of smart grids is Advanced Metering Infrastructures (AMIs). They consist of heterogeneous network communication-enabled devices that are structured hierarchically [12]. In this thesis, we assume they are structured as in Figure 2.1, however, the components can be structured in other ways. The lower layer consists of Smart Meters (SMs) that measure, collect, and forward data to either an optional intermediate level called Concentrator Units (CUs) or directly to the utility's Head-End (HE) [6]. Each SM can be connected to exactly one CU, whereas each CU can be connected to multiple SMs. The CUs collect consumption readings from the SMs and forward them to the utility's HE, which usually consists of central servers. The HE can be connected to the SMs directly or through CUs [12].

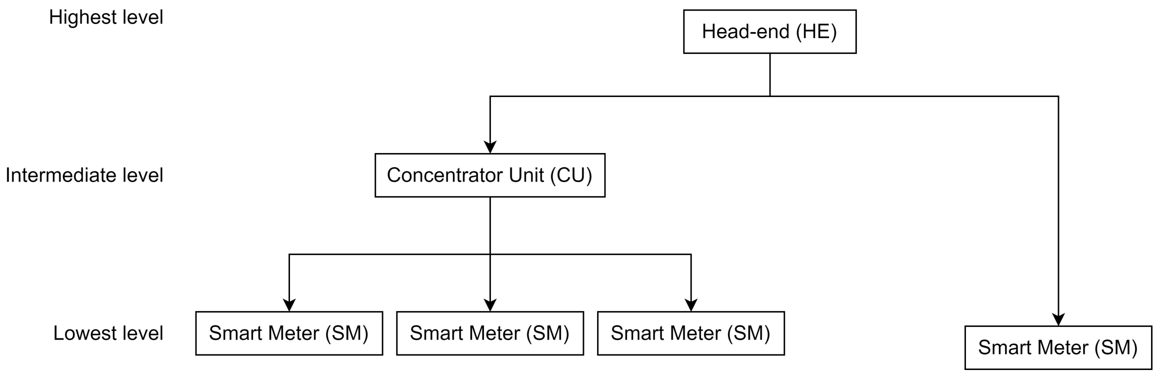

Figure 2.1: How different components in an AMI are organized.

A given set of data analysis operations over AMI data can be placed in different ways over the computational nodes found in such an AMI, for instance centrally at the HE. Data from SMs is sent to the utilitys HE. The HE has a more computational capacity than SMs which makes it possible for HE to handle large volumes of data and have access to all available data. A disadvantage of this approach is that all data needs to be moved to the HE. The more data that is moved, the longer it will take for the HE to process it, which increases the latency. In addition, the HE is bounded by its processing capacity. Therefore, other approaches can be of interest that let the meters perform some analysis on their own, as they have limited computation power and access to their own data. This could include, for example, discarding incorrect data. The approach can improve performance as it reduces the volume of data that is transferred in the system [6].

2.2 Stream processing

As stated in the Introduction, stream processing is a technique that continuously performs queries on unbounded data. This can be an option to handle the large volume of data that is exchanged in AMI. A stream S consists of a continuous, non-ending flow of incoming data. We define this data as tuples t_i. This means that a stream can be expressed as the following:

$$S = t_1, t_2, t_3, ...$$

Each tuple t within one stream S shares the same schema $\langle ts, a_1, a_2, ..., a_n \rangle$. The first part of the tuple ts, represents its creation time, whilst the second part of the tuple $a_1, a_2, ...a_n$ represents n application-related attributes [6]. Figure 2.2 shows a simplified example of a schema with tuples containing electricity usage readings from different meters in an AMI.

<ts, Meter ID, Hourly Consumption (kWh)>

```
<09:00, M1, 23.4>
<09:00, M2, 79.6>
<10:00, M1, 57.1>
```

Figure 2.2: Examples of tuples containing electricity readings.

Stream processing allows for queries on a continuous flow of incoming data. They are composed of sources, operators, and sinks. The source component is responsible for forwarding a stream of source tuples from a variety of sources to one or more operators. These operators can then manipulate the tuples or produce a new one with a different schema. Results from operators are eventually delivered to one or more sinks, which deliver results to end-users or other applications [8].

The stream processing paradigm is implemented through frameworks called Stream Processing Engines (SPEs). They make it possible to process a continuous flow of data in parallel and across different nodes [8]. One example of such an application is Apache Flink [13]. It was originally a research project that later became an Apache project in 2014 and is today used in production by several companies, including GE

[14]. This SPE and its DataStream API is used throughout the project because GE, GeneaLog [7] and Ananke [8] use it.

2.2.1 Operators

All major SPEs support user-defined operators and provide some standard operators that can be divided into stateless and stateful types. Figure 2.3 shows an illustration of all the operators that will be used in this thesis. Stateless operators do not maintain a state that develops according to the tuples being processed. Every operator instance processes one tuple at a time. The standard ones that SPEs provide are:

Map Applies a user-defined function to all tuples in a stream and returns exactly one tuple for each ingested tuple. The function can select one or more attributes from the input tuple. Each output tuple will have the same creation time ts as the input tuple.

FlatMap Another variant of the map operator with the difference that this returns arbitrarily many tuples for each input tuple, zero or more.

Filter Applies a user-defined filtering condition to all tuples in a stream to decide whether a certain tuple should be forwarded or discarded.

Union Takes two or more streams that share the same schema and merges them into one stream.

SPEs also provide stateful operators that depend on multiple input tuples and require that they be inside a window. The standard ones that SPEs provide are:

Aggregate Takes one or more tuples and combines them (possibly incrementally) through a custom function into one tuple. Use case examples can be the calculation of max, min, or sum.

Join Combines tuples from two different input streams with possible different schemas. The pair of tuples satisfying a user-defined condition will produce one output tuple through a custom function.

A query consists of operators that are connected through a Directed Acyclic Graph (DAG), where vertices represent operators and edges specify how tuples flow between them and are eventually delivered [8]. An example of a query that filters a stream based on `MeterID` and then removes a field from the tuples is illustrated in Figure 2.4.

2.2.2 Windows

Stateful operators like Aggregate and Join, mentioned in the previous section, require a mechanism to define non-blocking analysis over unbounded streams. One such method that is widely adopted is referred to as windowing. A window splits an unbounded stream into finite chunks and allows computation to be applied to the contents of the window. There are several types of windows and configurations for them [8].

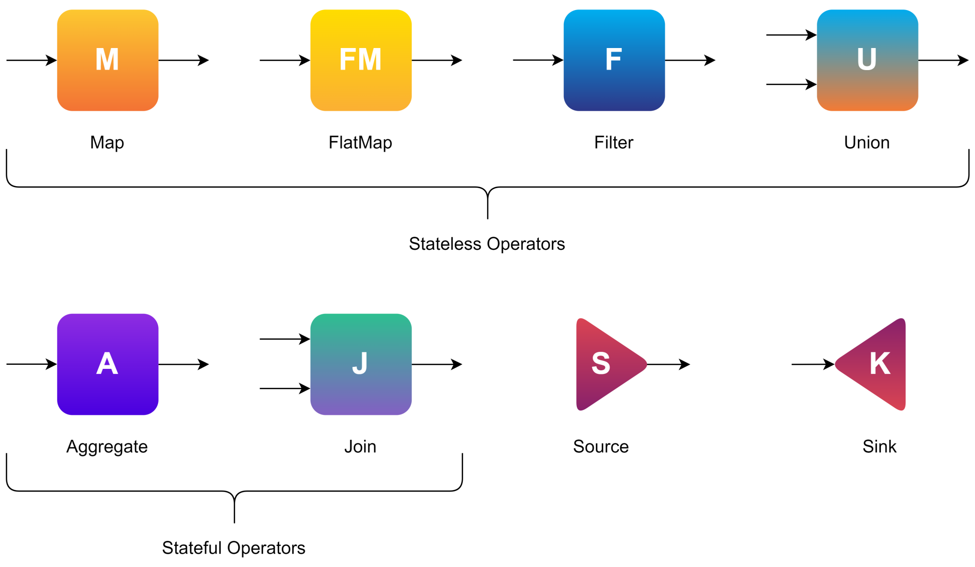

Figure 2.3: How operators, sources, and sinks will be illustrated in the thesis.

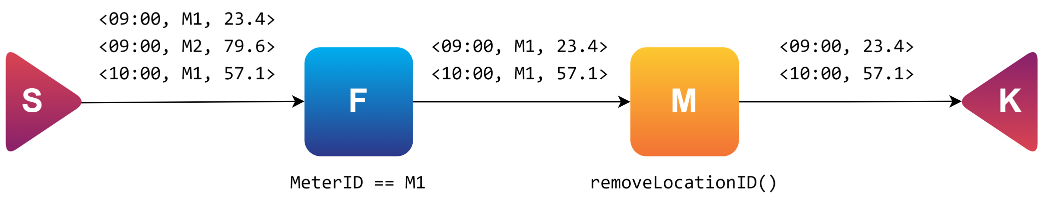

Figure 2.4: Example of a query that uses a filter and map operator.

A window can either be keyed or non-keyed. A keyed window splits the stream based on a key, which can be one or more attributes in a tuple. Every tuple in each window shares a common key. In the electricity reading example, the stream could be keyed on the attributes `MeterID`, creating several windows where the tuples in each of them share the same `MeterID`, see Figure 2.5. Compared to non-keyed windows, keyed windows have the advantage of allowing window computations to be performed in parallel as each window is independent from the rest [15].

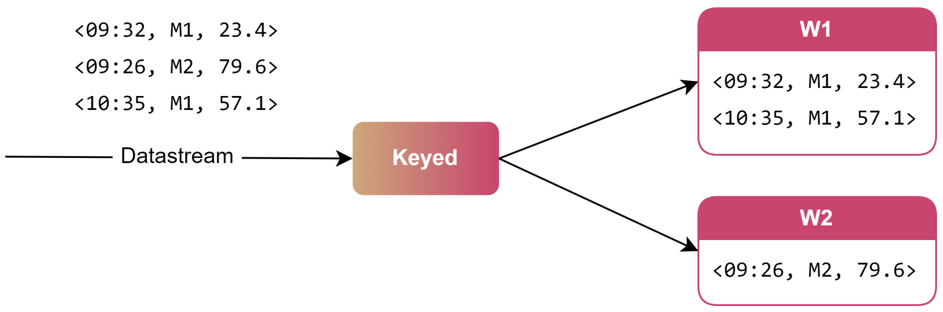

Figure 2.5: Keyed windows.

All windows need to be closed. A closed window indicates that no more tuples will be added and that results can be emitted. There are different ways to close a window, based on whether such a window is either count- or time-based. Count-based windows measure progress by the number of tuples. A window is marked complete when a certain number of tuples have been assigned to it. Time-based windows, on the other hand, progress by the timestamp ts contained in each tuple t. Windows are marked complete once their end time has been exceeded [15].

A window is created as soon as the first tuple arrives that should belong to the window. There are different window assigners that assign tuples to windows. This thesis uses two different ones, *Tumbling Window* and *Sliding Window*.

Tumbling windows

Tumbling window divides the stream into windows that are fixed size, non-overlapping, and contiguous time intervals. Each tuple can belong to at most one window. Time-based tumbling windows have a start- and end-time and also a fixed Windows Size (WS) [15]. For example, if WS is set to one hour and the start time is `09:00`, this will give the following windows: `09:00-09:59`, `10:00-10:59`, etc. As shown in Figure 2.6 tumbling windows do not overlap.

Sliding windows

Sliding windows are very similar to tumbling windows as they both have a WS. However, sliding windows also have an additional parameter called Window Advance (WA), which determines how frequently a window is started. Windows can therefore be overlapped, which can lead to tuples belonging to multiple windows [15]. Figure 2.7 shows an example of a time-based sliding window where WS is set to 2 hours and WA is set to 1 hour. This means that a new window will be started every hour with

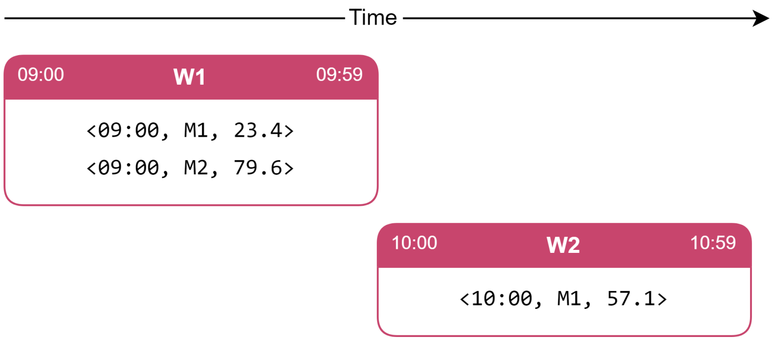

Figure 2.6: Tumbling windows.

the size of 2 hours. Notice that tuple `<10:00, M1, 79.6>` belongs to both window W1 and W2.

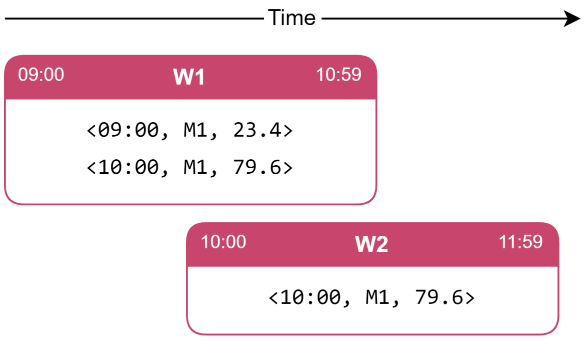

Figure 2.7: Sliding windows.

2.2.3 Watermark

A watermark in stream processing is a fundamental mechanism that helps the system keep track of time and deal with incoming tuples that are not in order (out of sequence). This is common in stream processing as the application is often run in parallel and distributed across several nodes. A watermark is a timestamp W_{ts} that tells the system about progress in time based on timestamps from incoming tuples. It is used when tuples arrive out of sequence to determine whether they will contribute to windows or operators. Watermark makes this possible by defining the lowest allowed time that an incoming tuple can have, which means that all tuples t with a timestamp t_{ts} that fulfill $t_{ts} \geq W_{ts}$ will be accepted while any tuple that fulfills $t_{ts} < W_{ts}$ will be discarded [8].

Figure 2.8 shows two different cases of a watermark strategy that increases the watermark in ascending order to guarantee strict order. In the first case, all tuples are received in order, which means that for all incoming tuples, $ts > W_{ts}$ will hold. The first tuple that arrives sets the watermark to `09:26`. The next tuple in the stream has a timestamp `09:32` which is greater than the watermark and will therefore be accepted. In this case, the watermark is also updated to `09:32`. The second case shows tuples arriving non-sequentially. The second tuple that arrives updates the watermark to `10:35` which will discard the third tuple with timestamp `09:32` as it is lower than the watermark. However, it is possible to add a maximum allowed latency to the watermark to handle streams that are out of order. The SPE will then place tuples in a buffer until the watermark has passed its timestamp.

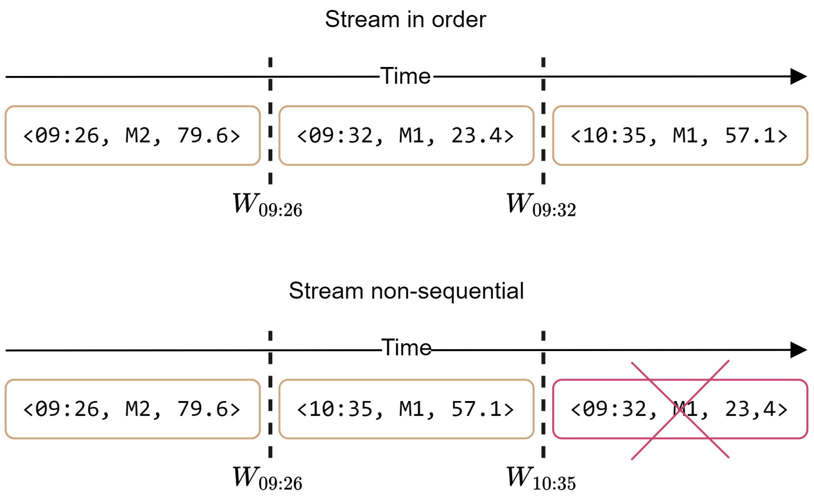

Figure 2.8: Illustration of two cases with a watermark strategy that guarantees strict order

2.3 Streaming provenance

Provenance in databases has been studied for several decades; however, there has been little research about provenance in stream processing as it is relatively new. There are several challenges with using provenance in a streaming application. For example, data is unbounded, making it impossible to have access to all data at once and thus have a global view like databases.

As mentioned in the introduction, stream provenance makes it possible to track back output in a stream to its source, as well as all the intermediate results from the source to the output. This can be realized by attaching annotations to the sink tuples that collect metadata about their origin and the production process [10]. One approach that both GeneaLog [7] and Ananke [8] use is to replace operators with modified ones that create and propagate annotations while producing regular data tuples. GeneaLog maintains a DAG called contribution graph for all sink tuples. It connects all source tuples that contributed to a sink tuple [7]. The graph for the query in Figure 2.4 is illustrated in Figure 2.9. The output can be easily verified by retracing the red arrows.

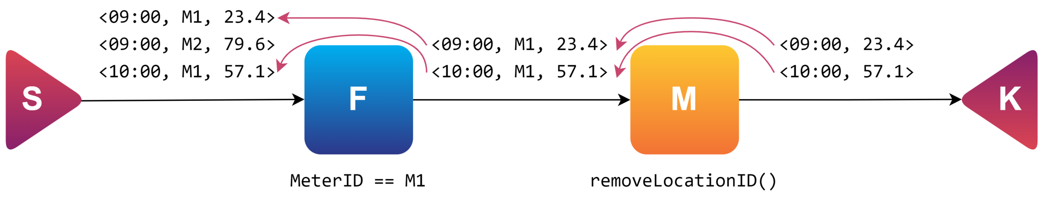

Figure 2.9: A query with provenance enabled.

3 Problem Definition

This chapter describes Göteborg Energi's current streaming application that generates forecasts. The problem with this system is explained and the requirements the new application should fulfill in order to solve it. Several questions are presented that the thesis aims to answer and lastly, some performance metrics of interest are defined that the evaluation should measure.

3.1 System overview

Göteborg Energi (GE) currently uses stream processing in a software that makes forecasts of future meter readings in their grid. For each meter, several forecasts are made based on historical data. The software makes it possible for an expert to view both the forecast and the outcome load. The application is used to balance the load on their grid by offering financial incentives to customers (often big companies) that temporarily lower their consumption during peak hours.

Figure 3.1 shows an overview of GE's forecast software and its different components. They use an internal Meter Data Management System (MDMS) to collect all readings from the meters in the grid. This data, together with weather forecasts from the Swedish Meteorological and Hydrological Institute (SMHI), is forwarded to Apache Kafka. That is a distributed publish-subscribe data streaming platform that can publish, subscribe to, store, and process streams of records in real-time. In GE's software, it is used to store the data and forward it to Apache Flink, which is the Stream Processing Engine (SPE) that GE uses [16]. It is used to calculate the electricity forecast, summarize the electricity consumption, and forward the result to be stored in a database for a limited time. One Flink module sums the electricity consumption readings and writes the result back to both Kafka and a database management system called InfluxDB. This is done so that other Flink modules can access the result. The data in Kafka is unsorted whereas the data in Influx is sorted based on time. The forecast Flink module uses the previous summation from the other Flink module to calculate the load forecast for each meter in the grid and, like the other one, stores the result in both Kafka and InfluxDB. Information about maximum allowed power consumption and grid status is stored in a database from the Network Information System (NIS). The data is visualized in a data visualization tool.

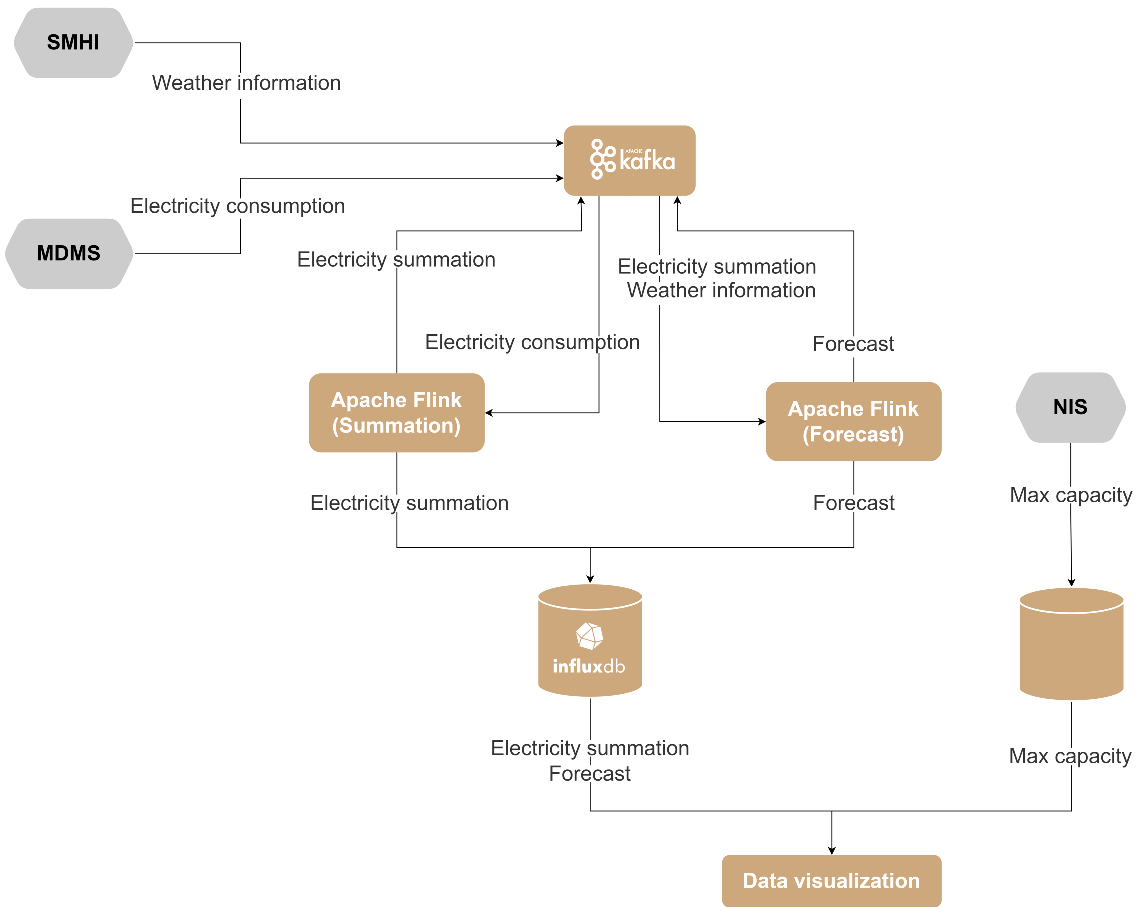

Figure 3.1: System architecture over GE's forecast system.

3.2 Forecast module

As presented in the previous section, GE uses a stream processing application that is run in real-time to generate forecasts and saves them temporarily in Kafka and InfluxDB, see Figure 3.1. For each hour the forecast module generates forecasts one week ahead of time, for each meter and hour in the week. This means that the forecast module will each hour generate 168 forecasts for each meter. To clarify this, an example is illustrated in Figure 3.2. If the current time and date are 9:00 01/05, then the forecast application will make its first forecast for 09:00 07/05 and set the prediction horizon to 168, stating that 168 hours before 09:00 07/05 it predicted the load. For the next hour, 10:00 01/05, all predictions will be made, and it will have 168 predictions.

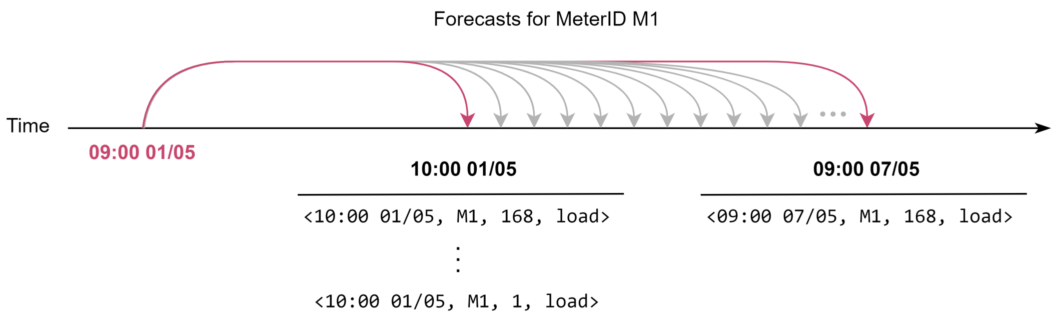
Forecasts for MeterID M1
Time
09:00 01/05
10:00 01/05
09:00 07/05
<10:00 01/05, M1, 168, load>
<10:00 01/05, M1, 1, load>
<09:00 07/05, M1, 168, load>

Figure 3.2: Example of how GE's forecast application generates forecasts.

3.3 System problem

In GE's system, the live data can sometimes have major deviations from the forecast, requiring careful handling by a human supervisor. Predictions made closer to the outcome time are more accurate than predictions made far ahead of time. Therefore only forecasts with a prediction horizon of 1 are of interest as the other forecasts will naturally have more deviation from the outcome reading. In some cases, deviation is expected, for example, when scheduled maintenance on the grid is performed or when a large production facility is started or stopped. In other cases, they are not, such as when there is an unexpected fault or maintenance work that has not been announced or scheduled. This will affect future forecasts by making them unreliable as they will be based on incorrect data.

Given the system in Section 3.1 we want to develop a prototype extension that is able to automate the classification and correlation of major divergence points with unknown faults to help data analysts detect meters that can be faulty. The extension should be able to detect large deviations between the forecast and the outcome load as this can be due to a faulty meter. However, it can be challenging to distinguish between a bad forecast and a faulty meter. If the deviation is due to a known fault, the extension should make the forecast module use past, and correct data instead. The extension can from the NIS find if there are any reported faults on the meter.

If there is no known fault reported, the extension should generate an alert for the meter that it is suspected to be faulty and make the forecast use old error-free data instead.

Sometimes it can be interesting to know the raw data that contributed to the alert. For example, if one reading from a meter is very high but the others are close to the forecast, that does not have to imply that the meter is faulty, just that one reading was very high. Streaming provenance can help recognize these kinds of problems. However, previous studies have shown that it requires more memory and causes latency compared to not using it [17]. Therefore, it is of interest to integrate and evaluate how streaming provenance can affect such an extension and if the advantages outweigh the weaknesses of using it.

3.4 Research questions

Based on the described problem and aim given in Section 1.3, the following questions are expected to be answered at the end of the work:

- How does provenance affect a stream processing application in terms of performance?
- Are the semantics provided by modern SPEs enough to inspect alerts and reports?

3.5 Performance metrics

In order to be able to determine how much provenance affects the application in terms of performance, several metrics of interest were chosen. They were the following:

CPU The average percentage of the total CPU time the application utilizes across all available processors.

Latency The average delay from a contributing source tuple arriving to a sink tuple being produced.

Memory The average amount of RAM the Stream Processing Engines (SPE) utilizes.

Throughput The average number of source tuples a query ingests per unit of time.

3.6 Ethical considerations

This master's thesis required detailed access to customers' electricity consumption, which, if not handled properly, could reveal sensitive information about the customer's life, such as when they sleep, wake up, go to bed, and so on. Sensitive information needs to be handled carefully to not leak outside of GE. Therefore, the prototype extension takes this into consideration and makes it impossible for external parties to access this information by avoiding using frameworks and APIs with known vulnerabilities.

4 Methodology

This chapter describes the design of the application and the technical implementation of it. In addition, the chapter also explains how provenance is implemented in the application and how the evaluation is performed.

4.1 Design

The prototype extension is connected to Göteborg Energi's (GE's) current system through their current database management system, InfluxDB because it is already in use. The extension is marked in pink in Figure 4.1.

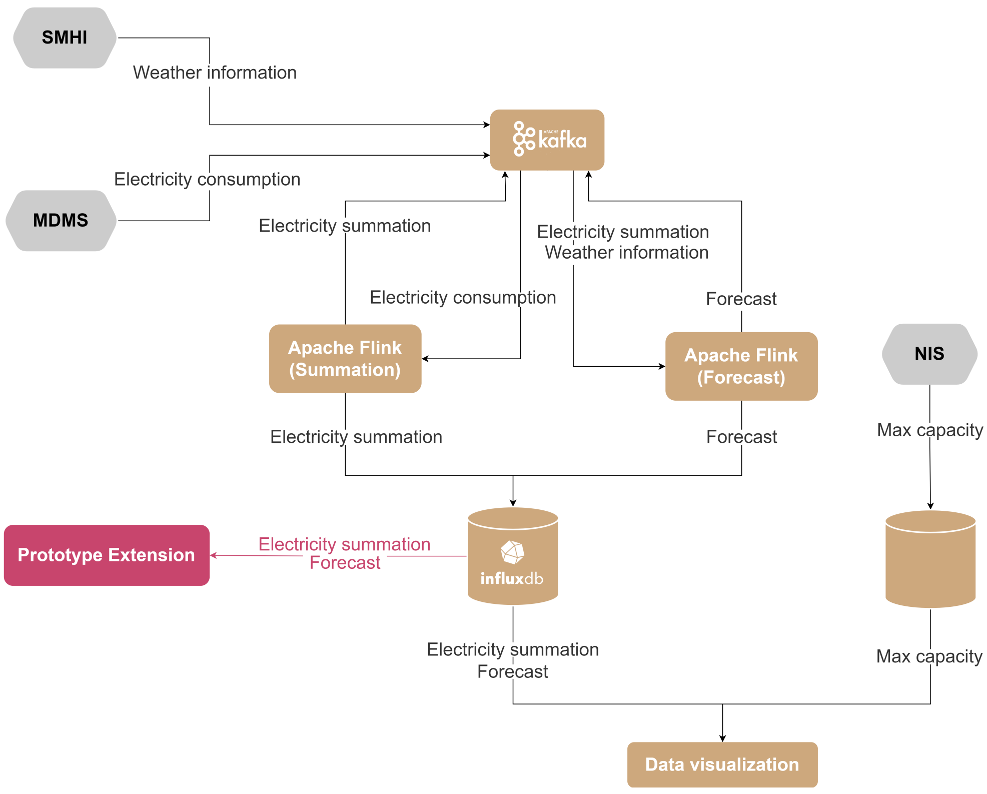

Figure 4.1: Placement of the prototype extension in GE's current system.

Two separate CSV files are downloaded from InfluxDB. One file contained the forecast data and the other one contained the electricity summation. These files are both formatted and sorted before the prototype application can read the data. The reason behind this is to reduce the waiting time in the extension and ingest data as they would in a real-life setting.

4.2 Implementation

4.2.1 Sort function

The downloaded files from InfluxDB are sorted by two additional small Flink applications. They both extract the values of interest from the files and create a list of tuples. These tuples are then sorted in two different ways. The electricity summation is sorted by registered time in increasing order. The forecast data is sorted based on $t_{ts} - t_{ph}$, where t_{ts} is the time the forecast is for and t_{ph} is the prediction horizon. By sorting the tuples in this order, the application receives tuples in a realistic order, as presented in Section 3.2. After the list has been sorted, it is saved in two separate files that the Flink application reads from through its source operators.

4.2.2 The application

The prototype extension in Figure 4.1 is implemented as a streaming application through a query. The query automatically calculates a maximum value of how much the meter reading can differ from the forecast. This limit value is calculated based on historical forecasts and updated regularly. A trigger is generated when a meter reading is exceeding the limit value as this can indicate a faulty meter.

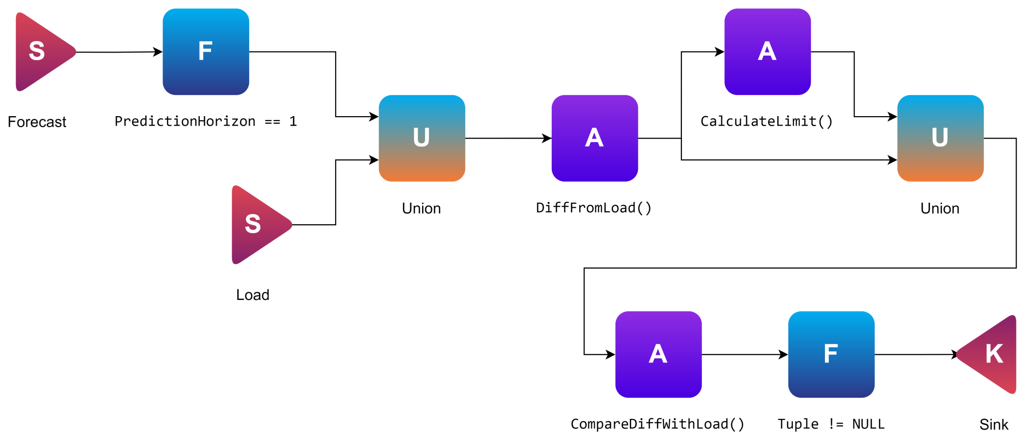

Figure 4.2: The query that the prototype extension used

Figure 4.2 shows the finalized query that is used in the project. The load and forecast source produce tuples that share the same schema `<`t_{ts}`, MeterID, Prediction Horizon, Load>`. The forecast stream is filtered based on `PredictionHorizon` == 1 to detect a sudden change. This can be changed to include more tuples without modifying other parts of the query or functions. Tuples are combined into one stream with a union operator. The stream is then split up based on timestamp t_{ts} and `MeterID` into tumbling windows of size 1 hour. The streams then enter an aggregation operator called `DiffFromLoad()` which performs two tasks. Firstly, it filters out incomplete pairs of tuples where either the load or forecast is missing. Secondly, it calculates the mean of the difference between forecast and load in percent. Exactly one new tuple with the schema <t_{ts}, `MeterID, DiffValue, IsLimit`> is produced for each correctly grouped tuple. The `IsLimit` field is set to false by the aggregation function because the produced tuples do not contain limit values. The stream is duplicated into two, one that is directly inserted into a union operator and the other one is split up based on `MeterID` into a sliding window of size 7 days and window advance 1 day, before entering the next aggregation operator. This aggregation operator is called `CalculateLimit()` and calculates a new limit value each day based on data from the previous 7 days. The tuples have the same schema as the previous aggregate operator but with the difference that `IsLimit` field is set to true. `CalculateLimit()` uses the same algorithm to calculate limit values as box plot outliers [18]. A simplified version of the algorithm is presented in Listing 4.1. The algorithm calculates a maximum difference between forecast and registered load based on previous forecast data. The duplicated stream with all differences and the other with limit values are then combined into one stream by the union operator and grouped based on `MeterID`. This stream is then inserted into tumbling windows of size 1 day before entering an aggregation operator called `CompareDiffWithLoad()` that compares all tuples with their corresponding limit tuple and generates a sink tuple with a trigger when $diff > limit$ is fulfilled. The `IsLimit` field in the tuples is used to differentiate between limit values and tuples containing derivations. The aggregation operator produces empty tuples when there are no triggers because an aggregation operator always has to return a tuple. The empty tuples are removed by a filter operator that is placed before the sink.

Listing 4.1: Algorithm to calculate limit value in CalculateLimit().

```
list = sort(list)
q1 = median(list(0, list.size/2)
q3 = median(list(list.size/2, list.size)
iqr = q3-q1
return q3+(1.5*iqr)
```

The two ingested data streams, load, and forecast, used two different watermark strategies. The load stream uses a watermark strategy that assumes the data is ingested in increasing order. The watermark W updates when a new tuple t arrives. The watermark updates to the maximum timestamp from either the incoming tuple or the previous watermark, $W_n = max(t_{ts}, W_{n-1})$. The watermark strategy for the forecast stream assumes the data is ingested in the order they are produced as described in Section 4.2.1. The watermark strategy updates the watermark when a new tuple is ingested with `PredictionHorizon` set to 1. The watermark is updated to the maximum timestamp from either the incoming tuple or the previous watermark, $W_n = max(t_{ts}, W_{n-1})$.

4.2.3 Implementation of provenance through Ananke

A copy of the query in Figure 4.2 is created, which is modified to use provenance. This project uses Ananke [8] to add provenance to the query. There are two versions of Ananke: ANK-1, which supports user-defined operators, and ANK-N which supports native operators, as presented in Section 2.2.1. In this thesis, ANK-N is chosen because it offers transparent implementation by encapsulation of operators.

Transparent implementation does not require the code to inherit provenance-specific code. Instead, encapsulation can be used to encapsulate each Flink operator with the appropriate framework decorators. To make this more clear, the corresponding code of the example given in Figure 2.4 is presented in Listing 4.2 and also how it looks with ANK-N in Listing 4.3. However, according to the researchers behind Ananke, ANK-N will increase the data serialization overhead and, as a result, also lower performance [8]. Despite this, encapsulation is chosen as it allows faster implementation of provenance in the query because few changes need to be made to an already working query.

Listing 4.2: An example of a simple query.

```
sourceStream
    .filter ( MeterID == "M1" )
    .map ( new removeLocationID() )
```

Listing 4.3: How provenance has been implemented to the example query presented in Listing 4.2 through encapsulation using ANK-N

```
ANK.source ( sourceStream )
    .filter ( ANK.filter ( MeterID == "M1" ) )
    .map ( ANK.map ( new removeLocationID() ) )
```

4.3 Evaluation setup

The evaluation aims to evaluate the application that is developed in the project. The evaluation compares the application with provenance enabled and disabled to find out how much provenance affects it.

4.3.1 Dataset

The data that is used to evaluate the extension is from real meters and stored in a database. The data is downloaded and sorted as described in Section 4.2.1, before being entered into the query. This makes sure that both variants are tested on the same data which makes a fair comparison and reproducibility possible. Six months of historical data from GE's 10 smart meters is used. The sizes of the two ingested streams' load and forecast differ significantly and are presented in Table 4.1.

Table 4.1: The size of the two ingested streams which are six months of historical data.

Stream	**Size**	**Number of tuples**
Load	143 963 bytes	3 282
Forecast	29 megabytes	592 402

4.3.2 Setup

The evaluation is performed on a remote server that is provided by GE. The server is accessed through Secure Shell (SSH). The server used 16 gigabytes of RAM and two CPUs called *Intel Xeon CPU E5-2695 v2 2.40GHz*. It runs an operating system called *Red Hat Enterprise Linux Server 7.9*.

100 iterations of tests are made on both versions of the query, the one with and the other without provenance. In order to do this and to measure the metrics described in Section 3.5 a modified version of the `run.sh` script from the Ananke [8] and GeneaLog [7] projects is used. The `run.sh` script automatically starts and stops each version of the query and performs a specified number of iterations of tests. In addition, it measures the metrics of interest and writes them into a file together with the sink tuples from the query.

The `run.sh` script collected data about CPU and memory consumption by sending HTTP requests to Flink. The latency of the tuples is calculated by letting tuple classes extend the `BaseTuple` class that holds a variable called `stimulus` which contains the time the tuple is ingested. This is then compared to the time when the corresponding sink tuple is produced. The throughput metric is measured by counting the number of lines read per second by the source.

5

Evaluation

This chapter presents and discusses the result of the evaluation of the two variants of the application. Each measured metric has a corresponding box plot, histogram, and table.

Figure 5.1 - 5.4 shows the generated plots for each measured metric. In each figure, the above graph is a box plot and the graph below is a histogram that shows the frequency of the values.

5.1 CPU

The box plot in Figure 5.1 shows that the CPU utilization is significantly higher when provenance is enabled. The histogram shows that the utilization is very high or close to the one where provenance is not used. Table 5.1 contains the exact values for the box plot.

Table 5.1: Summary of CPU utilization (%) in the tests.

	Median	Q_1	Q_3	**Min**	**Max**
No provenance	7.62	5.97	12.24	0.95	38.61
Provenance	25.73	7.21	39.27	0.98	43.29

The result showed that the provenance variant increased CPU utilization by 238.11%. This is much more overhead than any previous research had found [8], [17], [19]. There can be several reasons for this. One could be that the query's ingested data set is smaller than other similar researchers like Taube et al. [17]. They used a data set that is 1000 times larger than the one used in this thesis and had CPU utilization for both variants that on average is over 90%. A reason for the high CPU utilization could be the large data set that is used as the Stream Processing Engine (SPE) had a high volume of data to work with during evaluation. This could therefore leave less space for the two variants to have differences. Because this thesis used a much smaller data set, the CPU utilization never reached full capacity. The maximum CPU utilization in this thesis is 43%, which left more space for the two variants to have a difference.

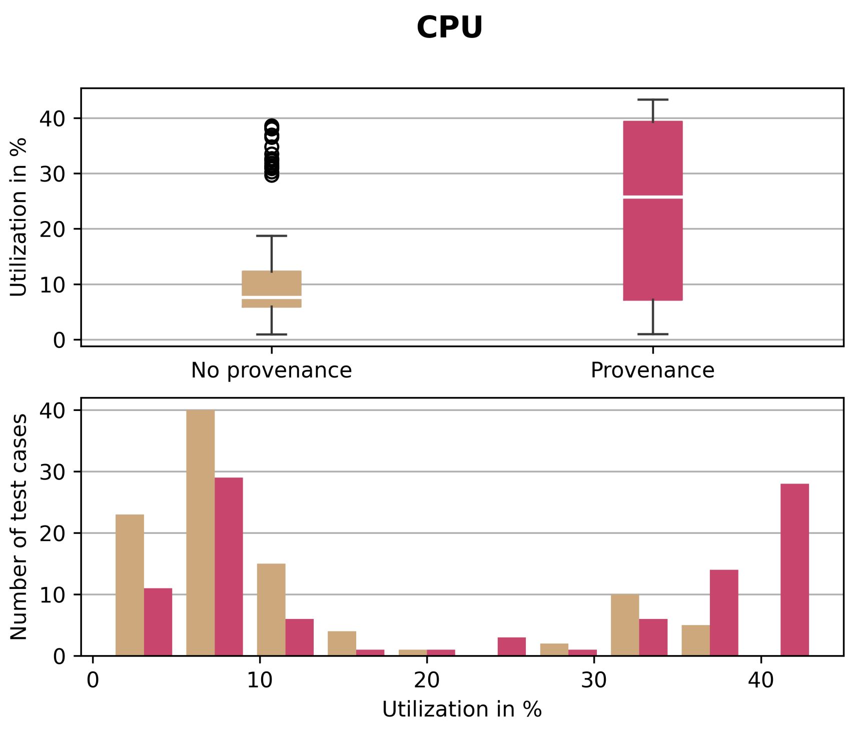

Figure 5.1: Box plot and histogram over CPU utilization during the application evaluation.

5.2 Latency

The box plot in Figure 5.2 shows that both variants took a similar amount of time to produce a sink tuple after an ingested source tuple. However, the histogram shows that the latency for the provenance one is slightly higher.

Table 5.2: Summary of latency (seconds) in the tests.

	Median	Q_1	Q_3	**Min**	**Max**
No provenance	3.25	3.02	3.49	2.41	5.73
Provenance	3.61	3.45	3.88	3.07	4.4

Table 5.2 shows that the latency is increased by 10.4% which is similar to other research [8], [17], [19]. This number is acceptable for the type of application that is developed as it does not require fast deliveries of triggers. However, for other types of applications like airbag systems in cars, this number can be considered too high. The latency in the tests could have been higher if the ingested data was larger because the query uses several stateful operators with large window sizes that buffer a lot of data.

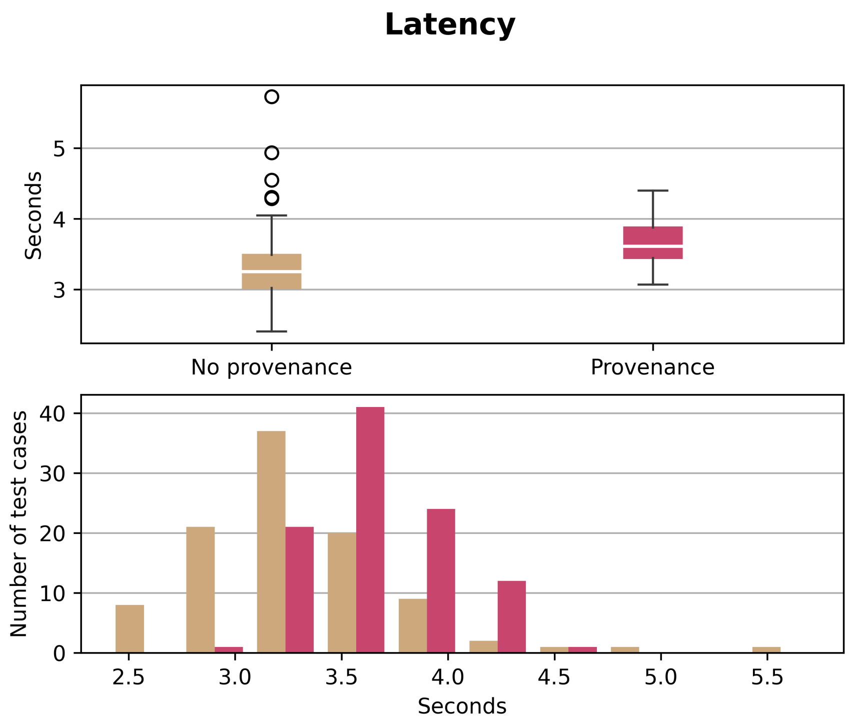

Figure 5.2: Box plot and histogram over latency during the application evaluation.

5.3 Memory

The memory consumption is shown in Figure 5.3. The box plot shows that the non-provenance variant consumes less memory than the provenance one because the third quartile (Q_3) is larger. However, the median in both are pretty close to each other.

Table 5.3: Summary of memory consumption (megabytes) in the test.

	Median	Q_1	Q_3	**Min**	**Max**
No provenance	230.97	227.21	240.98	221.88	325.1
Provenance	251.24	233.34	307.16	222.43	346.1

Table 5.3 shows that the memory consumption is increased by 8.78% when provenance is enabled which is in line with other research [8], [17], [19]. This is still a big increase for the type of streaming application that is developed as the number of smart meters can easily be increased. This thesis used data from 10 smart meters, but in real life, an Advanced Metering Infrastructure (AMI) can consist of thousands or millions of smart meters [20]. This can therefore require more memory storage when provenance is enabled.

It was expected for the application to consume more memory than it did because the query uses large window sizes and buffers the tuples in the aggregation functions. This could in turn make Ananke store more provenance data. There are several

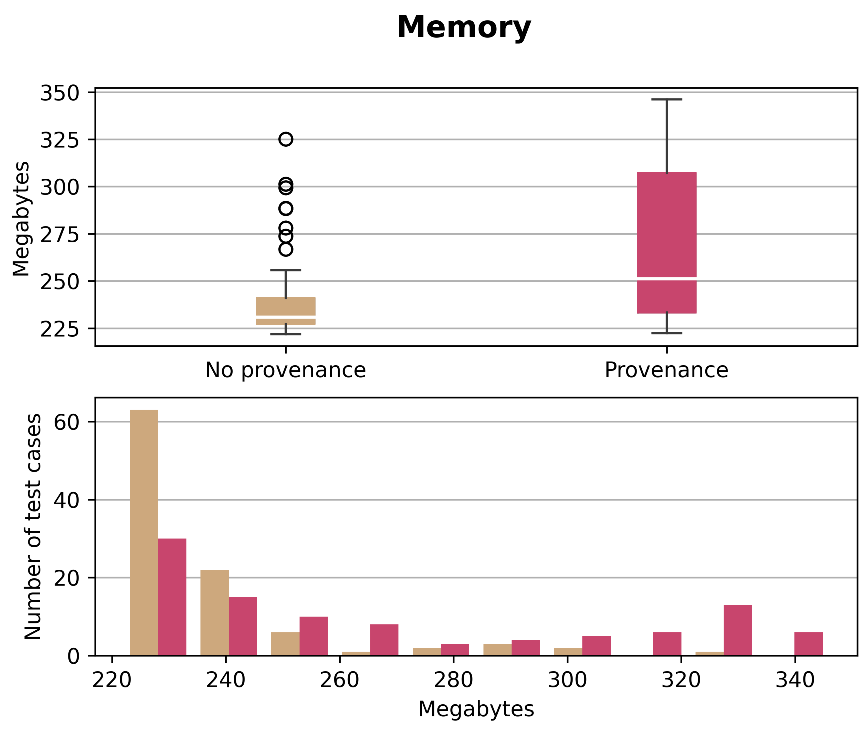

Figure 5.3: Box plot and histogram over memory consumption during the application evaluation.

reasons why the memory consumption is not more for the one with provenance. One could be the first filter function in the query that heavily reduces the data volumes and does not produce much provenance data.

5.4 Throughput

Figure 5.4 shows that the throughput of tuples for both variants is pretty close to each other. Table 5.4 shows that the throughput of tuples decreased by 10.4% when provenance is used. This is similar to other research [8], [17], [19]. The first and last measured values in each test are ignored because they deviate significantly from the rest of the measured values due to warm-up and cool-down mechanics in the SPE.

Table 5.4: Summary of throughput (tuples per seconds) in the test.

	Median	Q_1	Q_3	**Min**	**Max**
No provenance	118 300	105 134	129 487	59 487	185 714
Provenance	106 603	93 452	117 437	59 694	152 998

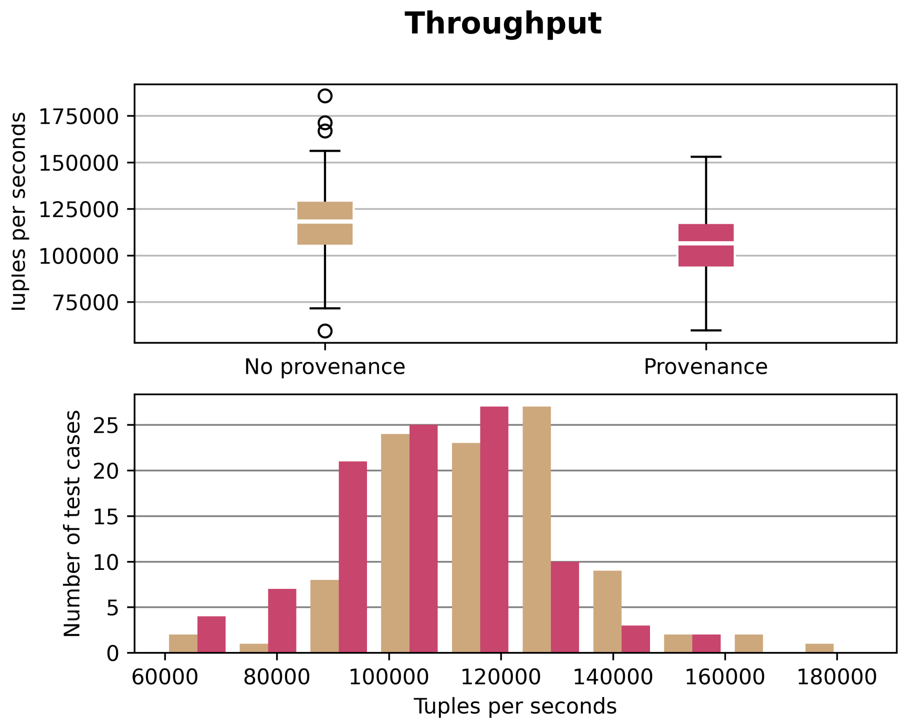

Figure 5.4: Box plot and histogram over throughput of tuples during the application evaluation.

6 Discussion

This chapter contains a discussion on the development process, faced challenges, limitations of the project, some similar projects, and lastly how it can be further developed.

6.1 Development process

6.1.1 Apache Flink

There were several challenges and problems when developing the two variants of the application. One challenge with developing a streaming application with Apache Flink is the new concepts, like variables being used to contain both static values and streams. For example, usually when an application is created that finds the max value, a variable is defined that contains the current max value when iterating through the data. However, a stream processing application requires a stateful operator like aggregate that keeps a state of the current highest value in the stream.

Another challenge is the limited information about Flink and how to use it. This made the development processes challenging and took more time than originally planned. There are not enough web pages or research papers with information about it. The only major source with information is Flink's official web page and the API documentation, which is not always sufficient. Additionally, Flink is regularly updated with new versions which can sometimes differ significantly from other versions. This can make the documentation obsolete fast and Stack Overflow posts irrelevant.

6.1.2 Data ingestion

In one version of the application, data is ingested from Kafka. A major problem with this approach is that there is no control over the data in Kafka. The data is constantly changing as new values are added by the meters. This should have affected the evaluation of the application and made an unfair comparison as different tests could be run with different data. Besides this, there are also problems with duplicated and unsorted data. Because data are unsorted, this approach required the application to buffer large amounts of forecast data and wait before the watermark is passed. Therefore data is ingested from Influx instead.

6.1.3 Ananke

The transparent version of Ananke is used to implement provenance to the query. There is a trade-off using this version as the research paper [8] showed that performance decreased compared to the non-transparent version. However, the implementation is very easy and fast, which in many cases reduces development and maintenance costs, but it can also be costly in the long run as more resources are consumed by it.

Adding provenance to the query is difficult as there is no complete documentation on how to implement Ananke into an already working query. For example, it is not obvious which changes need to be made to an already working query. Instead, several examples in the repository are used to figure out how to implement it. Another challenge with Ananke is that not all operators were supported. This delayed the development phase as the query needed to be rewritten several times because it used unsupported operators. In addition, it also forced the application to use less efficient solutions. For example, originally in one version of the query the last aggregation operator (`CompareDiffWithLoad()`) and filter operator in Figure 4.2 was a RichFlatMap operator. That operator kept the limit value as a state and updated it every time a new limit tuple arrived. The other tuples were then compared to the current limit state and triggers were sent out whenever a value exceeded the limit, just like the current aggregation operator `CompareDiffWithLoad()`. This approach was more efficient as no tuples needed to be buffered in a window. However, because Ananke did not support RichFlatMap, other solutions needed to be used. Another alternative could have been to use ANK-1, which supports user-defined operators. This should make it possible to use the solution with RichFlatMap and in the end have a more efficient query. However, due to time constraints, this option is not possible to implement. This is left for future research to implement and evaluate.

6.2 Limitations

The result obtained from this project has some limitations that should be taken into account. They will be listed in the following sections.

6.2.1 The data

The data that is used in the evaluation had a fixed size as it is downloaded from InfluxDB. This meant that all data is available at once which can be considered a limitation because this is not the case in real life. Instead, in this case, data is ingested by the smart meters every hour or at other intervals in the future. However external delays like data availability should be eliminated to evaluate and make fair comparisons. Otherwise, there is a risk of test cases taking unnecessarily long time and making the result heavily dependent on the rate of the ingested data.

Another limitation can be the ingested data set used in this thesis, which can be considered small compared to other similar studies. A different or more clear result could have been obtained if the data set was larger. Another problem with this is that several outliers were introduced in the result, see box plots in Figures 5.1 - 5.4. They need to be considered when interpreting the result.

It is worth pointing out that the evaluation uses all available data from Göteborg Energi's (GE's) database. However, due to unknown reasons, a large portion is missing. The files from InfluxDB had significantly less data than would be expected from 6 months' worth. In addition, many load tuples did not have matching forecasts and vice versa. The solution could have been to use data from more smart meters or generate synthetic data to fill out the missing ones.

6.2.2 Optimization

The query and the aggregation functions used in the project were created firsthand to fulfill the goal, namely to generate triggers for suspected faulty meters. The application is therefore not optimized in terms of time complexity and performance due to time constraints. This can be considered a limitation as the application could performed better if it had been optimized. The Ariadne [10] project showed that optimized query reduces overhead for large window sizes and has less computation time and latency.

Another limitation could be that the thesis uses the transparent version (ANK-n) of Ananke, which according to the researchers behind it consumes more resources than the non-transparent version (ANK-1) [8]. Therefore the provenance variant of the application could have performed better and had less overhead if the non-transparent version is used.

6.2.3 Other frameworks

This project used Ananke [8] to implement provenance to a query made in Flink. The main reason behind this is that GE already had a system that used Flink and support from the authors of Ananke is possible. However, there are several other popular Stream Processing Engines (SPEs), like Apache Spark and Apache Storm [21] that could be of interest to study. Other frameworks that offer provenance to a streaming application like Ariadne [10] could also be of interest. Other results could have been obtained if these SPEs and frameworks were used.

6.3 Related work

While provenance can explain why a given outcome is produced, it does not help to determine why an expected outcome is missing which can make it difficult to validate and debug streaming queries. Therefore tools that can help understand why an expected outcome in a stream is missing, are of interest. The framework Erebus [22] solves this by allowing users to define boolean expectation predicates on the results of a query, verifying at run-time if such expectations hold, and also providing explanations when expected and observed outcomes diverge (missing answers). The framework is built on top of Apache Flink and can be used with provenance tools like Ananke [8], GeneaLog [7] and Ariadne [10].

Several research studies have been conducted on the security aspect of Advanced Metering Infrastructure (AMI) that uses stream processing to create an intrusion detection system [6], [23]–[25]. There has also been research made on data analysis on smart grids through Apache Spark [26].

6.4 Future work

As presented in the introduction stream applications and stream provenance are still fairly new and not as well established as batch processing. This thesis work shows some challenges and opportunities with developing and using stream provenance in a real-life setting, in this case in an AMI. In addition, the performance loss with using provenance for the application is studied. These findings can be used by developers and researchers as guidelines when developing stream-based applications that use provenance. The study can also be used to further develop Ananke and make it more efficient and easier to use.

This thesis did not cover the security aspect of provenance. This is particularly interesting when the application is distributed across several nodes on a public network as it can be targeted for attacks. An example of a question of interest can be, "Does the use of provenance in a distributed data stream application have any security vulnerabilities?"

Apart from the question presented in Section 3.4 there are other questions of interest that could not be covered in this thesis due to time restrictions. One is regarding the accuracy of triggers: how many of the generated triggers are due to a faulty meter, and how many are false? Another question is regarding experts' use of the application: how can the developed application leverage experts knowledge to steer classification and correlation?

The aggregation operator `CalculateLimit()` in Figure 4.2 currently uses the algorithm described in Listing 4.1 to calculate the limit values. In order to calculate more efficient limit values, machine learning models could be used to make classification of the data. This is out of scope for this thesis, but it is suggested for future research to implement this.

Due to time restrictions, it is not possible to make the application use data from the Network Information System (NIS), see Figure 3.1. Therefore even if a smart meter is reported to be faulty the application still generates triggers for it and the forecast application uses faulty data for future forecasts which makes the forecast less accurate. It is suggested that future developers make the application use data from the NIS to avoid these kinds of problems.

The result showed that the CPU utilization is significantly higher when provenance is enabled. This result is different from other research. Several factors could yield this result, one of which involved the amount of ingested data. The same is true for memory consumption, as the values were more spread and no concrete conclusions could be drawn from the result. It can therefore be interesting to know how much these metrics change when the size of the ingested data set and the number of tests is varied.

Lastly, in the previous section, several limitations were presented. These limitations should also be addressed in future research.

7
Conclusion

The increased volume of data that Advanced Metering Infrastructure (AMI) entails can be solved by using stream processing. The lost data traceability in stream processing can be solved by using provenance. However, provenance is understudied in the context of stream processing and how performance is affected by it. Göteborg Energi (GE) currently uses stream processing to make forecasts of future meter readings in their grid. However, sometimes this forecast differs significantly compared to the actual outcome reading and this can be due to faulty meters.

This thesis has successfully developed a stream-based application in Apache Flink which makes use of a framework called Ananke [8] that offers provenance to find sink tuples' origin. The application can read data from smart meters in an AMI to automatically calculate reasonable limit values based on the ingested data and generate alerts when meter readings exceed the limit value. Flink allows the application to be distributed across several nodes to handle the increased data volume from AMI. The application can be used with provenances and without, to compare the pros and cons of using it.

The result achieved from evaluating the application is similar to other research, except for the increased CPU utilization, which is significantly higher than other has shown. There can be several reasons for this increased value. The likely one is that other studies used a bigger data set which caused a high degree of CPU utilization when provenance is both enabled and disabled. This leaves less gap for differences. Therefore it is suggested that future research varies the volume of the ingested data set and observes how the different metrics change.

For future work, it is recommended to also eliminate factors that caused some uncertainty in the project's results. Namely, to use a bigger data set, try the application in a real-life setting, let experts evaluate its usefulness, study the generated triggers, try to optimize the application, and connect it to GE's Network Information System (NIS) to make the application ignore already reported faulty meters. Before provenance can be used in a public application, the security aspect of it needs to be studied and possible vulnerabilities with using it need to be looked into. Apart from this, other Stream Processing Engines (SPEs) and stream provenance frameworks are also suggested to be used and compared.

Bibliography

[1] T. museet. “Elektricitet.” (), [Online]. Available: https://www.tekniskamuseet.se/lar-dig-mer/100-innovationer/elektricitet/ (visited on 01/26/2023).

[2] Swedenergy. “Sveriges elbehov 2045.” (Feb. 2023), [Online]. Available: https://www.energiforetagen.se/globalassets/dokument/gap-rapport/sveriges-elbehov-2045---hur-stanger-vi-gapet-20230215.pdf (visited on 05/09/2023).

[3] G. W. Arnold, “Challenges and opportunities in smart grid: A position article,” *Proceedings of the IEEE*, vol. 99, no. 6, pp. 922–927, 2011. DOI: 10.1109/JPROC.2011.2125930.

[4] R. Jiang, R. Lu, Y. Wang, J. Luo, C. Shen, and X. Shen, “Energy-theft detection issues for advanced metering infrastructure in smart grid,” *Tsinghua Science Technology*, vol. 19, pp. 105–120, Apr. 2014. DOI: 10.1109/TST.2014.6787363.

[5] S. Riksdag. “Förordning (1999:716) om mätning, beräkning och rapportering av överförd el.” (), [Online]. Available: https://www.riksdagen.se/sv/dokument-lagar/dokument/svensk-forfattningssamling/forordning-1999716-om-matning-berakning-och_sfs-1999-716 (visited on 01/24/2023).

[6] V. Gulisano, M. Almgren, and M. Papatriantafilou, “Online and scalable data validation in advanced metering infrastructures,” in *IEEE PES Innovative Smart Grid Technologies, Europe*, 2014, pp. 1–6. DOI: 10.1109/ISGTEurope.2014.7028740.

[7] 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.

[8] D. Palyvos-Giannas, B. Havers, M. Papatriantafilou, and V. Gulisano, “Ananke: A streaming framework for live forward provenance,” vol. 14, no. 3, pp. 391–403, Dec. 2021, ISSN: 2150-8097. DOI: 10.14778/3430915.3430928. [Online]. Available: https://doi.org/10.14778/3430915.3430928.

[9] H.-S. Lim, Y.-S. Moon, and E. Bertino, “Research issues in data provenance for streaming environments,” in *Proceedings of the 2nd SIGSPATIAL ACM GIS 2009 International Workshop on Security and Privacy in GIS and LBS*, ser. SPRINGL ’09, Seattle, Washington: Association for Computing Machinery, 2009, pp. 58–62, ISBN: 9781605588537. DOI: 10.1145/1667502.1667516. [Online]. Available: https://doi.org/10.1145/1667502.1667516.

[10] B. Glavic, K. Sheykh Esmaili, P. M. Fischer, and N. Tatbul, "Ariadne: Managing fine-grained provenance on data streams," ser. DEBS '13, Arlington, Texas, USA: Association for Computing Machinery, 2013, pp. 39–50, ISBN: 9781450317580. DOI: 10.1145/2488222.2488256. [Online]. Available: https://doi.org/10.1145/2488222.2488256.

[11] G. Stad. "Göteborg energi." (), [Online]. Available: https://www.goteborgenergi.se/ (visited on 01/25/2023).

[12] 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, Hamilton, New Zealand: Association for Computing Machinery, 2018, pp. 171–182, ISBN: 9781450357821. DOI: 10.1145/3210284.3210298. [Online]. Available: https://doi.org/10.1145/3210284.3210298.

[13] T. A. S. Foundation. "Stateful computations over data streams." (), [Online]. Available: https://flink.apache.org (visited on 12/06/2022).

[14] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *The Bulletin of the Technical Committee on Data Engineering*, vol. 38, no. 4, 2015.

[15] A. S. Foundation. "Windows." (), [Online]. Available: https://nightlies.apache.org/flink/flink-docs-master/docs/dev/datastream/operators/windows/ (visited on 04/09/2023).

[16] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink : Stream and batch processing in a single engine," *The Bulletin of the Technical Committee on Data Engineering*, vol. 38, no. 4, 2015, No ISSNQC 20161222. [Online]. Available: http://sites.computer.org/debull/A15dec/issue1.htm.

[17] J. Taube and W. Johnsson, *Streaming analytics with provenance in the advanced metering infrastructure*, 2022. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=ir01625a&AN=cst.20.500.12380.305852&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds.

[18] GeeksforGeeks. "Box plot." (Jan. 2021), [Online]. Available: https://www.geeksforgeeks.org/box-plot/ (visited on 10/16/2023).

[19] A. Erlandsson and M. Gordani Shahri, *Interactive fine-grained provenance for streaming-based analysis applications*, 2021. [Online]. Available: https://odr.chalmers.se/server/api/core/bitstreams/2861b9e1-f5f4-4e82-a5e9-5b45229cb6cb/content.

[20] JRC Smart Electricity Systems. "Smart metering deployment in the european union." (), [Online]. Available: https://ses.jrc.ec.europa.eu/smart-metering-deployment-european-union (visited on 10/17/2023).

[21] 6sense. "Best stream processing software in 2023." (2023), [Online]. Available: https://6sense.com/tech/stream-processing (visited on 08/05/2023).

[22] 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.

[23] M. A. Faisal, Z. Aung, J. R. Williams, and A. Sanchez, "Data-stream-based intrusion detection system for advanced metering infrastructure in smart grid: A feasibility study," *IEEE Systems Journal*, vol. 9, no. 1, pp. 31–44, 2015. DOI: 10.1109/JSYST.2013.2294120.

[24] C. H. Park and T. Kim, "Energy theft detection in advanced metering infrastructure based on anomaly pattern detection," *Energies*, vol. 13, no. 15, 2020, ISSN: 1996-1073. DOI: 10.3390/en13153832. [Online]. Available: https://www.mdpi.com/1996-1073/13/15/3832.

[25] N. Chu, A. Williams, R. Alhajj, and K. Barker, "Data stream mining architecture for network intrusion detection," in *Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration, 2004. IRI 2004.*, 2004, pp. 363–368. DOI: 10.1109/IRI.2004.1431488.

[26] Shyam R., B. Ganesh H.B., S. Kumar S., P. Poornachandran, and Soman K.P., "Apache spark a big data analytics platform for smart grid," *Procedia Technology*, vol. 21, pp. 171–178, 2015, SMART GRID TECHNOLOGIES, ISSN: 2212-0173. DOI: https://doi.org/10.1016/j.protcy.2015.10.085. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2212017315003138.