Thesis for The Degree of Doctor of Philosophy

Explainable and Resource-Efficient Stream Processing Through Provenance and Scheduling

Dimitrios Palyvos-Giannas

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2022

Explainable and Resource-Efficient Stream Processing Through Provenance and Scheduling

Dimitrios Palyvos-Giannas

ISBN 978-91-7905-692-6
Doktorsavhandlingar vid Chalmers tekniska högskola
Ny serie nr 5158
ISSN 0346-718X

Technical Report No 222D
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 (0)31-772 1000

This thesis has been prepared using LaTeX.
The cover image was designed using resources from Flaticon.com
(users kerismaker, Freepik).

Printed by Chalmers Digitaltryck
Gothenburg, Sweden 2022

"O Beautiful the shaking of heads
Over the indisputable truth!"

— Bertolt Brecht
In Praise of Doubt, 1932

Explainable and Resource-Efficient Stream Processing Through Provenance and Scheduling

Dimitrios Palyvos-Giannas
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

In our era of big data, information is captured at unprecedented volumes and velocities, with technologies such as Cyber-Physical Systems making quick decisions based on the processing of streaming, unbounded datasets. In such scenarios, it can be beneficial to process the data in an online manner, using the *stream processing* paradigm implemented by Stream Processing Engines (SPEs). While SPEs enable high-throughput, low-latency analysis, they are faced with challenges connected to evolving deployment scenarios, like the increasing use of heterogeneous, resource-constrained *edge* devices together with *cloud* resources and the increasing user expectations for usability, control, and resource-efficiency, on par with features provided by traditional databases.

This thesis tackles open challenges regarding making stream processing more user-friendly, customizable, and resource-efficient. The first part outlines our work, providing high-level background information, descriptions of the research problems, and our contributions. The second part presents our three state-of-the-art frameworks for explainable data streaming using *data provenance*, which can help users of streaming queries to identify important data points, explain unexpected behaviors, and aid query understanding and debugging. (A) *GeneaLog* provides *backward provenance* allowing users to identify the inputs that contributed to the generation of each output of a streaming query. (B) *Ananke* is the first framework to provide a duplicate-free graph of *live forward* provenance, enabling easy bidirectional tracing of input-output relationships in streaming queries and identifying data points that have finished contributing to results. (C) *Erebus* is the first framework that allows users to define expectations about the results of a streaming query, validating whether these expectations are met or providing explanations in the form of *why-not* provenance otherwise. The third part presents techniques for execution efficiency through custom *scheduling*, introducing our state-of-the-art scheduling frameworks that control resource allocation and achieve user-defined performance goals. (D) *Haren* is an SPE-agnostic user-level scheduler that can efficiently enforce user-defined scheduling policies. (E) *Lachesis* is a standalone scheduling middleware that requires no changes to SPEs but, instead, directly guides the scheduling decisions of the underlying Operating System. Our extensive evaluations using real-world SPEs and workloads show that our work significantly improves over the state-of-the-art while introducing only small performance overheads.

Keywords: Stream Processing, Data Streaming, Provenance, Scheduling

Acknowledgment

I want to thank my supervisor, Vincenzo Massimiliano Gulisano, for always being there during the past five years and having complete trust in me, even when we disagreed. Thank you, Vincenzo, for having an open mind, taking the time to listen, and being understanding when needed. I would also like to thank my co-supervisor, Marina Papatriantafilou, for her patience, insightful suggestions, and constant presence during my Ph.D. Many thanks to all people I collaborated with, and especially to Gabriele Mencagli and Katerina Tzompanaki for our very productive work together. I especially acknowledge my master thesis supervisor, Dimitris Fotakis, for encouraging me to pursue a Ph.D. and helping me in the first steps toward this goal. Furthermore, I thank Angelos Arelakis and everyone at ZeroPoint for the fruitful internship.

I am honored to have professor Peter Pietzuch as the faculty opponent of the defense and professors Seif Haridi, Matthias Weidlich, Evangelia Kalyvianaki, and Per Stenström on the grading committee. I also want to thank my Ph.D. examiner, Gerardo Schneider, for his support and guidance.

Thank you, Georgia, Christos, Thomas, Hannah, Valentin, and Fazeleh, for always being there to talk and forcing me to take much-needed breaks from work. Special thanks go to Ivan, Yiannis, and Iosif for their invaluable guidance during all steps of my Ph.D. Also, thank you, Bastian, for the excellent collaboration and the engaging discussions with our two other officemates, Karl and Romaric. Also, thanks to Aras, Babis, Oliver, Petros, Stavros, and Vaggelis for the fun beers after work. Additionally, I would like to thank all the people I interacted with in the corridors, classrooms, and lunches, for making the workplace a fun place, including (but certainly not limited to): Adones, Ahmed, Ali, Aljoscha, Amir, Bei, Beshr, Carlo, Elad, Fransisco, Huaifeng, Joris, Kim, Kåre, Magnus, Nadja, Nasser, Olaf, Pedro, Philippas, Prajith, Rolf, Shiliang, Uddipana, and Yiannis Sourdis. Moreover, I would like to thank all the administrative staff that went above and beyond to help, Clara, Eva, Jenny, Marianne, Monica, Rebecca, Lars, and Michael, as well as Tomas, the head of my unit, who was always available to answer my questions.

Last but not least, I want to thank the people who made all this possible. My fantastic parents, who made countless sacrifices to ensure that I could follow my dreams. My lovely sister, who was always there to ~~bother~~ support me. My amazing friends in Sweden (little Christina, Dimitris, Eva, Hamdy, Manos, Maria, Suvi, Vasiliki, Linn, Linnea, Lovisa, to name a few), Greece (too

many to name here), and beyond, for supporting me and filling my free time with great moments. And, of course, I thank my beautiful partner, Pinelopi, for her love, support, and unlimited patience during this journey.

Funding sources. This work was supported by the Swedish Research Council (Vetenskapsrådet), project *"HARE: Self-deploying and Adaptive Data Streaming Analytics in Fog Architectures"*, with grant number 2016-03800.

Dimitrios Palyvos-Giannas
Gothenburg, August 2022

List of Publications

Appended publications

This thesis is based on the following publications:

[A] **D. Palyvos-Giannas**, V. Gulisano, and M. Papatriantafilou
"GeneaLog: Fine-Grained Data Streaming Provenance in Cyber-Physical Systems"
Parallel Computing, vol. 89, p. 102552, 2019.

[B] **D. Palyvos-Giannas**, B. Havers, M. Papatriantafilou, and V. Gulisano
"Ananke: A Streaming Framework for Live Forward Provenance"
Proceedings of the VLDB Endowment, vol. 14, no. 3, pp. 391–403, 2020.

[C] **D. Palyvos-Giannas**, K. Tzompanaki, M. Papatriantafilou, and V. Gulisano
"Erebus: Explaining the Outputs of Data Streaming Queries"
Under revision.

[D] **D. Palyvos-Giannas**, V. Gulisano, and M. Papatriantafilou
"Haren: A Framework for Ad-Hoc Thread Scheduling Policies for Data Streaming Applications"
Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems, 2019, pp. 19-30.

[E] **D. Palyvos-Giannas**, G. Mencagli, M. Papatriantafilou, and V. Gulisano
"Lachesis: A Middleware for Customizing OS Scheduling of Stream Processing Queries"
Proceedings of the 22nd International Middleware Conference, 2021, pp. 365–378.

Other publications

The following publications were published during my PhD studies, or are currently in submission/under revision. However, they are not appended to this thesis, due to contents overlapping that of appended publications or contents not related to the thesis.

[a] **D. Palyvos-Giannas**, M. Papatriantafilou, and V. Gulisano
"Research Summary: Deterministic, Explainable and Efficient Stream Processing"
Proceedings of the 2022 Workshop on Advanced tools, programming languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems, pp. 65–69.

[b] M. Gordani Shahri, A. Erlandsson, **D. Palyvos-Giannas**, and V. Gulisano
"Poster: Twins, a Middleware for Adaptive Streaming Provenance at the Edge"
International Conference on Distributed Computing and Networking 2021, pp. 235–236.

[c] V. Gulisano, D. Jorde, R. Mayer, H. Najdataei, and **D. Palyvos-Giannas**
"The DEBS 2020 Grand Challenge"
Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems, 2020, pp. 183–186.

[d] V. Gulisano, **D. Palyvos-Giannas**, B. Havers, and M. Papatriantafilou
"The Role of Event-Time Order in Data Streaming Analysis"
Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems, 2020, pp. 214–217.

[e] L. Bortolussi, V. Gulisano, E. Medvet, and **D. Palyvos-Giannas**
"Automatic Translation of Spatio-Temporal Logics to Streaming-Based Monitoring Applications for IoT-Equipped Autonomous Agents"
Proceedings of the 6th International Workshop on Middleware and Applications for the Internet of Things, 2019, pp. 7–12.

[f] **D. Palyvos-Giannas**, V. Gulisano, and M. Papatriantafilou
"Demo Abstract: Haren: A Middleware for Ad-Hoc Thread Scheduling Policies in Data Streaming"
Proceedings of the 20th International Middleware Conference Demos and Posters, 2019, pp 19–20.

[g] **D. Palyvos-Giannas**, V. Gulisano, and M. Papatriantafilou
"GeneaLog: Fine-Grained Data Streaming Provenance at the Edge"
Proceedings of the 19th International Middleware Conference, 2018, pp. 227–238.

[h] I. Walulya, **D. Palyvos-Giannas**, Y. Nikolakopoulos, V. Gulisano, M. Papatriantafilou, and P. Tsigas
"Viper: A Module for Communication-Layer Determinism and Scaling in Low-Latency Stream Processing"
Future Generation Computer Systems, vol. 88, pp. 297-308, 2018.

Research Contribution

I contributed to **Paper A, Paper C, Paper D, Paper E** as the lead designer and main implementor. I also led the writing of the manuscripts and collaborated with all other authors. In **Paper B**, me and Bastian Havers were the lead designers and main implementors.

Contents

Part I

Thesis Overview

1 Introduction

> *"The world's most valuable resource is no longer oil, but data"*
>
> — The Economist (2017) [1]

During the last decades, *data* has become an incredibly powerful force that affects, directly or indirectly, most aspects of our daily lives. This *data revolution* has enabled cutting-edge applications that seemed like science fiction a few years ago, such as recommendation engines that know what you want to buy before you even realize it, Smart Cities with traffic lights that can coordinate based on road conditions to reduce traffic jams and car emissions, and computers that can recognize human faces better than humans themselves [2–4].

Similar to other valuable resources such as oil, data is not very useful in its raw form. Instead, it needs to be refined or *processed* before value can be extracted out of it. However, data processing is becoming increasingly challenging due to the rapid increase in the complexity and volume of modern-day datasets, also referred to as *big data*. Big data materialized as a result of the widespread digitization of our society that expanded the number and variety of data sources and applications that continuously collect raw information about people and their environment, such as smartphones, social media, connected vehicles, and financial transactions [5, 6].

Looking back in time, in the 1970s researchers and industry were studying the concept of *database machines*, all-in-one solutions for storing and processing data, with a focus on robustness and performance [7]. A bleeding-edge database machine in the 1980s, called DBC/1012, could store and process one terabyte of data (one terabyte is equal to 10^{12} bytes, thus the machine's name) [8]. DBC/1012 was impressive for its time: it could store approximately fifty times the entire content found in the English Wikipedia in 2022 [9]. However, as

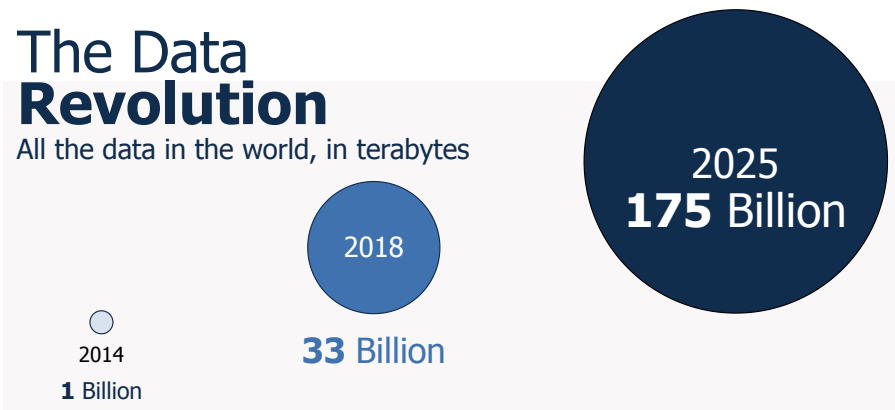

Figure 1: The data revolution in numbers. The area of each circle is proportional to the volume of all data in the world at that year.

visualized in Figure 1, the data gathered in the recent years is growing so quickly that, in 2014, the world would have needed *one billion* DBC/1012 machines to store all of its data (i.e., one zettabyte), with the figure growing to 33 billion in 2018 and 175 billion by 2025 [10]. This growth of modern datasets is not only in volume but also in velocity: in order to extract the most value out of the data, it must be collected and analyzed as quickly as possible [11]. For instance, traffic lights in a Smart City need to be able to react quickly to changing road conditions. Suppose the system that processes the data for such traffic lights responds with long delays. This could lead to situations where the environment has changed when the respective action takes place, causing potentially undesirable consequences.

Aiming to maximize the extracted value from today's rapidly growing datasets, we propose state-of-the-art solutions that make such processing more efficient and explainable and enable the next generation of novel data-centric applications. This thesis focuses on *stream processing* (also known as *data streaming*) [12–14], a processing paradigm ideal for today's high-velocity datasets, in which the processing happens in an online manner, as the data arrives, without having to wait for all of it to become available. We study how stream processing applications can transition from centralized, cloud-based deployments to edge architectures, which move part of the data processing closer to the end devices that collect and react to the data [15]. Such architectures can, on the one hand, bring benefits due to their proximity to the data sources and the users but, on the other hand, introduce challenges due to their heterogeneity and resource limitations. In the context of streaming queries deployed on heterogeneous edge and cloud nodes, we propose techniques that explain unexpected behaviors through *data provenance*, which can trace important inputs and outputs of such queries. Additionally, we explore custom *scheduling*, which can increase the efficiency and level of control over the available processing resources. As described later in the thesis, we design and implement novel frameworks that address pressing issues in data provenance and scheduling, and evaluate our work on real-world streaming applications.

In the remainder of this overview, we dive deeper into data processing and discuss different processing models (§2, §3, §4), present a short background of data provenance (§5) and scheduling (§6), outline open research problems, defining the research questions posed by the thesis (§7), and summarize the contributions made to address these questions (§8).

2 The Era of (Big) Data

Modern datasets are characterized by an *"increase in the volume of data that are difficult to store, process and analyze through traditional database technologies"* [5]. Previous works discuss four dimensions, also known as the *four V's* [5,6] of (big) data, which can be a useful way to characterize datasets and identify challenges associated with processing them:

(1) **Volume**: The size of the datasets is increasing dramatically due to an expanding array of new technologies and data sources.

(2) **Variety**: Present-day datasets come in heterogeneous formats, e.g., video, audio, text. Unstructured datasets are becoming increasingly common.

(3) **Velocity**: New data points are produced (and, thus, need to be processed) at a fast rate, frequently as data streams.

(4) **Value**: Significant value concealed inside the datasets needs to be extracted through processing that can keep up with the other three dimensions.

The exponential increase in data volumes has been fueled by the expansion of various data sources, including smartphones, environmental sensors, industry processes, transactions, video, and positional reports [3,16]. Two of the leading technologies connected to this dramatic increase in the volume of high-velocity data produced daily are the World Wide Web (or simply the web) and the Internet of Things.

Web-based social networking platforms such as Facebook, Twitter, Instagram, and YouTube provide the companies that own these platforms with vast amounts of data related to their users' interactions. For example, during every minute of 2019, more than 500 hours of new videos were uploaded to YouTube, on average [17]. Facebook, with its more than 2.32 billion users connected in a social graph with more than 140 billion friend connections, has had more than 219 billion photos uploaded to its systems [18,19]. These data volumes are only expected to increase: by 2025, people are expected to engage in digital services 4900 times per day (i.e., once every 18 seconds) [10].

The *Internet of Things (IoT)* is a rapidly growing paradigm that focuses on expanding the functionality of everyday devices such as fridges, washing machines, and bus stops by embedding sensors into them and giving them networking capabilities. Different from the web, such sensors usually require no manual user interaction and thus collect vast amounts of heterogeneous data such as positional reports, environmental measurements (e.g., temperature, humidity), health data (e.g., from smartwatches), etc. [11]. For instance, a modern vehicle can generate dozens of gigabytes per day [20], a jet engine can generate a terabyte per day [21] and an autonomous car more than 70 terabytes per day [10]. Estimates predict *1 trillion* active IoT sensors by 2030, which would make the IoT one of the largest (if not the largest) contributors to the volume of available big data. In contrast to other big data sources, IoT devices frequently produce *noisy* data, the majority of which can be uninteresting [11]. For example, imagine an IoT temperature sensor in a smart fridge that monitors the temperature inside the fridge every 30 seconds to verify it stays within the food safety limits. In the usual scenario, such a sensor will report a huge volume of regular temperature reports until an abnormal event that requires further action is observed (e.g., a component failure). Noisy IoT data introduces new data processing challenges, which are discussed later in this part.

2.1 How Does Data Create Value?

The vast amount of information created by and about individuals from the sources discussed above has been an enabler of a new generation of applications that are changing the daily lives of billions of people worldwide [5,6].

To begin with, data has been a significant enabler of the vast improvements in the capabilities of *Artificial Intelligence* (AI). Many now-popular AI algorithms were developed decades ago, but it was the availability of large training datasets (together with advancements in hardware capabilities) that

allowed such algorithms to be used in real-world scenarios [22]. The social and behavioral interaction data captured by web-based services, social networks, and IoT sensors has helped businesses make big leaps in fields such as image analysis, speech recognition, natural language processing, and recommendation systems [2] that have brought applications such as personalized advertising campaigns, voice-controlled virtual assistants, accurate machine translation services, and semi-autonomous vehicles that can recognize road characteristics and automatically respond to road hazards.

Big data is also revolutionizing our daily lives in less obvious ways, through its effect on the IoT and especially on *Cyber-Physical Systems (CPSs)*. The latter are physical systems that integrate computation with actions in the physical environment [23, 24]. Unlike traditional embedded systems, CPSs are not isolated devices but, instead, are designed around *networking* numerous sensors, actuators, and processing units to make intelligent decisions by utilizing the increased information exchange and processing capacity of such architectures [25]. CPSs can have ground-breaking applications in fields such as medicine, manufacturing, avionics, smart buildings, and critical infrastructure [24]. An example of a CPS is the expanding Smart Grid infrastructure, which can allow power companies to observe the electricity consumption of each customer in real-time [26–28]. Not only can such real-time measurements inform analysts about urgent events (e.g., a localized blackout), but they can also assist in large-scale projects and predictions, such as long-term planning for the infrastructure of the power grid. Notice that the Smart Grid is only a small part of a larger group of Smart City applications, which extend to many aspects of citizens' daily lives, including education, vehicle traffic control, and waste management [3]. Furthermore, in contrast to traditional embedded systems enclosed inside a single device, Cyber-Physical Systems can become enormous, stretching across cities, countries, or even the whole planet [29]. For instance, planet-scale social networks can act as Cyber-Physical Systems, having "social sensors" that can detect large-scale anomalous events through the interactions of the users in the social graph, as indicated by a research prototype that detected earthquakes by analyzing Twitter data [30].

3 Stream Processing

3.1 From Batch to Stream Processing

Since data is becoming *"too big, too fast, or too hard for traditional databases"* [31], researchers and practitioners have been experimenting with novel processing paradigms to extract the value out of such datasets. The two leading paradigms nowadays are 1) *batch processing*, which focuses on high-throughput processing of vast but *bounded* datasets, and 2) *stream processing*, which focuses on the processing of *unbounded* datasets with high throughput and low latency, and is the topic of this thesis.

Batch processing is a natural way for humans to process data. It involves waiting until all the necessary data is available and then executing the desired computation. People have experimented with (semi-)automated batch processing for more than 100 years: during the 1890 US Census, the Hollerith tabulating machines aggregated data such as age, gender, etc. from punched

cards, replacing the need for tedious manual counting that took a long time to complete [32]. Modern batch processing is represented by frameworks such as MapReduce [33], Hadoop [34], and Spark [35], which can handle the processing of petabyte-scale datasets by transparently parallelizing the computations on commodity hardware and reducing the complexity of issues such as fault-tolerance, load balancing, and data distribution. Batch processing frameworks are designed around the assumption that the datasets to be processed might be vast in volume, but they are bounded, i.e., the whole dataset is available in some persistent storage. Batch processing systems are usually executed in periodic intervals (e.g., every day or every month), with the processing taking hours to days to complete. While batch processing can extract value from large datasets with high throughput, its assumption about bounded datasets is problematic when analysts need to process unbounded datasets of streaming data and receive answers with low response times.

Stream processing (or *data streaming*) is a complement to batch processing that targets use cases where the data is unbounded [14]. For example, imagine a Smart Grid application that computes the average electricity consumption in a city, based on data collected by smart energy meters. The data reported by such meters is unbounded since there exists no point in time at which the dataset is complete: the meters continuously report the electricity consumption for as long as they are active. Batch processing systems could approach this problem by gathering a subset of the data, e.g., for each day, and then computing the desired average over only that subset. However, such an approach can introduce unacceptable high *latency* (i.e., delay in getting a result), which can make it impractical to make decisions based on the output of the data analysis. On the other hand, stream processing is designed from the ground up on the assumption that data is unbounded. Stream processing systems *continuously* process data as it arrives by *windowing* the data items based on the time of the corresponding event and offering the option for *incremental* computations. Because of this, stream processing can produce much lower latency results than its batch counterpart while potentially requiring fewer resources by not requiring all data to be loaded at once. In the Smart Grid example discussed above, an analyst could use stream processing to incrementally compute the average electricity consumption every minute (e.g., by maintaining two counter variables, one for the total consumption and one for the number of reports) without requiring the data to be persistently stored or available beforehand.

Stream processing has been studied since the 2000s with researchers implementing pioneering *Stream Processing Engines* (SPEs) such as Aurora and Borealis [12, 36]. Nowadays, academia and industry take advantage of the low-latency results offered by the data streaming model by actively developing new SPEs suited for modern computing architectures and datasets. Notable examples include Google DataFlow [14], Apache Flink [37], Facebook Puma, Stylus, and Swift [38], Microsoft Trill [39], Twitter Storm and Heron [38, 40], Apache Samza [41]. Though these SPEs can exhibit significant differences in their API, runtime architecture, query optimization, and deployment options, they are based on some fundamental shared assumptions of the stream processing model, outlined below.

3.2 Stream Processing Background

The work of this thesis builds on the DataFlow model [14] that is adopted by SPEs such as Apache Flink [37]. *Streams* represent unbounded sequences of *tuples*, each of which has a *timestamp* (i.e., its *event-tine*, which is the time when the event corresponding to the tuple happened) and a list of user-defined attributes. A *streaming query* is a Directed Acyclic Graph (DAG) of *Sources*, *operators* and *Sinks*. Sources (also referred to as *Ingress* operators) generate *source tuples* that correspond to events (e.g., from IoT sensors, mobile phones, social network interactions) and send them through *streams* to one or several operators. Operators process tuples using user-defined functions and can discard tuples and/or forward (potentially new) tuples downstream in the query DAG. Query results eventually reach the query Sinks (or *Egress* operators) as *sink tuples* and are sent to end-users or other applications.

SPEs come with *native operators* such as Filter, Map, Aggregate, and Join, which generally behave similarly to their relational database counterparts. SPEs also allow users to define *custom operators* if necessary. Operators can be *stateless*, processing one tuple at a time, or *stateful*, maintaining groups of tuples as *time windows* and computing the results based on the contents of such windows. SPEs ensure correctness even in parallel/distributed executions with potentially out-of-order input data through mechanisms such as *watermarks* [14], which bound on the degree of out-of-orderness present in each stream, or *sorting* the input tuples of each operator based on their timestamps [42].

Figure 2 shows an example streaming query that uses native operators. Events arrive at the query from external data sources and are converted to tuples by the query Source. In this example, the tuples are messages posted by social network users. The tuple attributes include the timestamp (τ), the user that posted the message (*user*), the message itself (*msg*), and the number of "likes" that the message received (*likes*). The query uses a Map operator to convert each message to lowercase (top stream) and a Filter to keep only users below 500 (bottom stream). Notice that the Map keeps the same timestamp and can change (some of) the other attributes, whereas tuples that match the Filter pass through it unaltered. After the Map, an Aggregate computes the average likes per message over all users for each hour. The outputs of the Aggregate are new tuples, with a timestamp (usually) equal to the end of the window they refer to. Then, a Join computes the ratio of likes of each message compared to the average over that hour, with the output timestamps being (usually) equal to the end of the Join's window. Finally, the query results arrive at the Sink, to be returned to the analyst or forwarded to another system.

When deploying a streaming query, SPEs transform the *logical DAG* of the query (i.e., the DAG that was defined by the user, also known as a *topology* [43]) of *logical operators*, to a *physical DAG*, of *physical operators*. The physical DAG is the entity executed by the SPE on the underlying machine, with the physical operators being the smallest execution units of the SPE. During the transformation from the logical to the physical DAG, the SPEs can apply optimizations such as *parallelism* (or operator *fission*) and *chaining* (or operator *fusion*) [44]. Since parallelism and chaining interact with several aspects of our research contributions, we discuss them in more detail below.

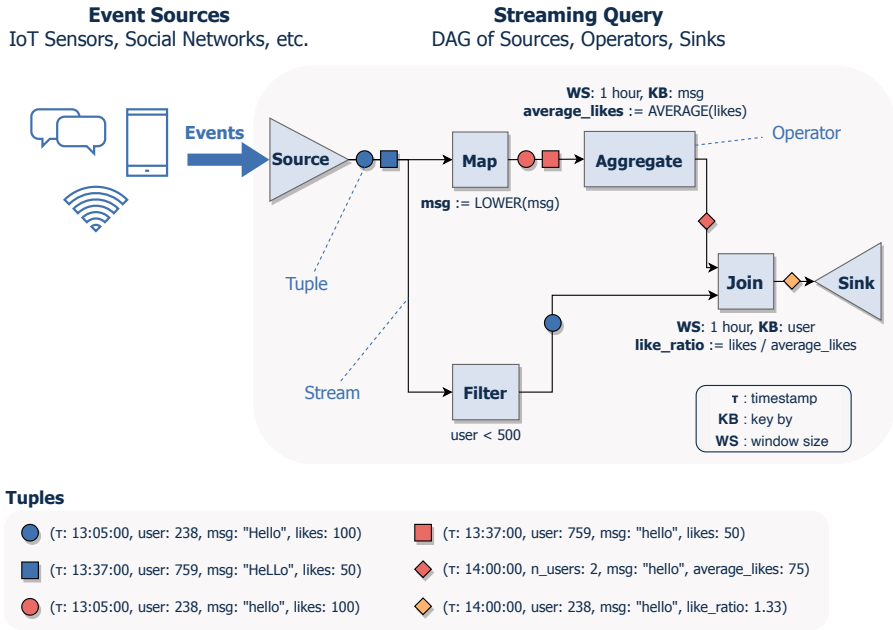

Figure 2: Example streaming query that finds how "liked" each user message is, compared to the average likes over the last hour for messages with identical content. Tuples are illustrated as shapes of different colors, with the detailed attributes of each tuple shown at the bottom of the figure.

Parallelism (Operator Fission) SPEs can deal with demanding workloads by utilizing several CPU cores and machines by *parallelizing* and *distributing* the computations, respectively. Parallel processing is achieved by splitting a computation into smaller parts and assigning each of these parts to a different computational unit (e.g., CPU core or processing node). In the field of data streaming, parallelism appears either as *task parallelism*, where different operators are executed in parallel, and *data parallelism* (i.e., operator fission), where multiple instances of the same operator are running in parallel to (usually) process different parts of the data. A particular case of task parallelism, frequently found in modern SPEs, is *pipeline parallelism*, where operators with a producer-consumer relationship are executed independently and in parallel [44].

These different forms of parallelism are shown in Figure 3. Figure 3a illustrates task parallelism for operators B and C, as well as the special case of pipeline parallelism. In data parallelism, shown in Figure 3b, there is a *splitter* responsible for deciding which parallel operator instance should receive each tuple (e.g., based on hashing), and a *merger* that merges the outputs of multiple instances, possibly enforcing ordering or watermark constraints. Depending on the implementation of the SPE, the splitter and merger can be separate runtime components or integrated inside the operator [45].

In this thesis, we are primarily interested in data parallelism, as this technique can help resolve *bottleneck operators*, whose processing speed cannot keep up with the arrival rate of their inputs. Such operators can activate an SPE-specific *backpressure* mechanism [37, 46] that slows down the rest of the

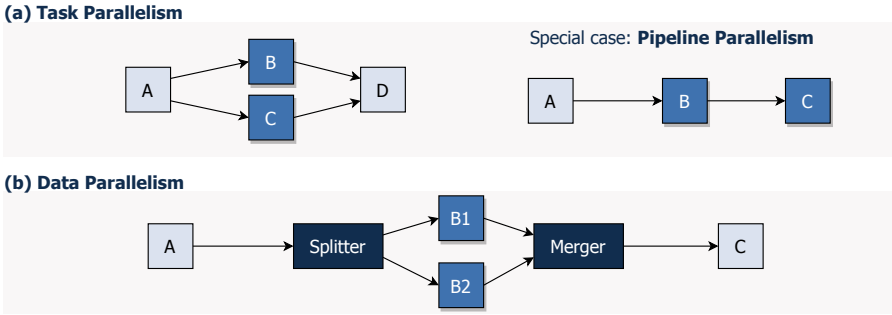

Figure 3: The different types of parallelism found in stream processing.

pipeline, leading to a degradation of the throughput and latency of the query. Data parallelism causes each parallel operator instance to process less data per time unit, allowing the operator to keep up with its inputs and preventing the activation of backpressure and any related performance degradation. Modern SPEs can transparently parallelize their native operators but might require guidance in the case of user-defined custom operators.

Chaining (Operator Fusion) When the processing performed by an operator is small, its communication and runtime costs might be higher than the operator's processing logic. In such cases, SPEs can perform operator fusion (also referred to as chaining) to improve query performance by merging lightweight logical operators in the same physical operator [44].

4 Cloud and Edge Computing

4.1 Cloud Computing

Cloud computing is a paradigm that aims to provide *"computing as a utility"* by making hardware and software resources available to anyone in a *"pay-as-you-go"* model [5, 47]. Cloud computing revolutionized data processing by removing the upfront costs related to purchasing expensive computing infrastructure and the running costs of maintenance and administration. Instead, a *cloud provider* rents out such infrastructure, with the user being charged only for the resources actually utilized. Cloud computing can be provided in various levels of abstraction, such as *Software As a Service (SaaS)*, which includes services such as Gmail and Dropbox, *Platform as a Service (PaaS)*, including services like AWS Lambda and Google App Engine, or *Infrastructure as a Service (IaaS)*, which includes services like AWS and Microsoft Azure. The three main hardware characteristics [47] of cloud computing are:

(1) The illusion of **infinite resources** that can be requested when necessary, which means that users of cloud services do not need to have long-term provisioning plans. This is different from traditional computing, where a company would need, e.g., to order and configure new servers a long time before they need to be up and running to support growing demand.

(2) The absence of any **upfront costs**, as the cloud users only pay for the resources they utilize, and only for as long as they are utilizing them.

(3) The ability to **scale on demand**, which allows a cloud user to quickly switch, for example, from using one machine to one thousand machines and pay a proportional price for the utilized resources, mitigating issues related to over- and under-provisioning.

In connection to parallel processing, we note that cloud computing usually offers *"cost associativity"*, where users are charged equally whether they use one machine for a thousand hours or a thousand machines for one hour, a feature that encourages and rewards the utilization of massively-parallel architectures, as the latter can deliver results much quicker for the same infrastructure cost.

4.2 Edge Computing

The increasing volume and velocity of data, with an expectation of 29 billion connected devices in 2022 [48] are challenging the cloud paradigm since, in many cases, it is neither feasible nor desirable to transfer all raw data to the cloud for processing as such communication round-trips can introduce non-trivial delays to the end applications [10]. Furthermore, especially in IoT applications, only a tiny fraction of that raw data might actually be important, so processing all raw data in the cloud can also waste significant amounts of network bandwidth with minimal increase in the extracted value [15, 49, 50]. Lastly, privacy and other regulations might make it difficult to get permission to transfer non-aggregated data to the cloud [51].

Edge computing (also known as *fog computing* [52] or *cloudlets* [53]) aims to mitigate the above issues by introducing an additional processing step between the data sources and the cloud, allowing the applications to move (part of) the processing to infrastructure closer to the devices and applications by taking advantage of the increasing computational capacity of *edge nodes* such as base stations, switches or routers, as well as edge devices such as smartphones and energy meters [15]. The edge refers to the fact that such devices are further away from the core of the network, i.e., the big data centers that comprise the cloud. Figure 4 outlines the layering of the cloud and the edge, including example devices of each layer. In general, as one moves further down the pyramid, the computational capacity decreases, but the proximity to the user increases, and thus the communication delays and costs decrease. For simplicity, in the following, we use the term *edge device* to refer to both edge nodes and edge devices, except if there is a need to distinguish between the two.

Edge devices are by definition close to the data sources, and thus they can reduce the response time and the bandwidth usage of data processing by either processing all the data locally or, if the analysis requires data from multiple devices, performing initial filtering and aggregations of the data before sending it to the next layer in the processing hierarchy (e.g., the cloud) for further processing [49]. Such performance improvements are critical enablers of Cyber-Physical Systems, as the lower response times make it possible to make timely decisions based on the (processed) sensor data [29]. Additionally, the physical proximity of edge devices to the data sources allows them to have *location awareness*, enabling exciting new features such as the detection

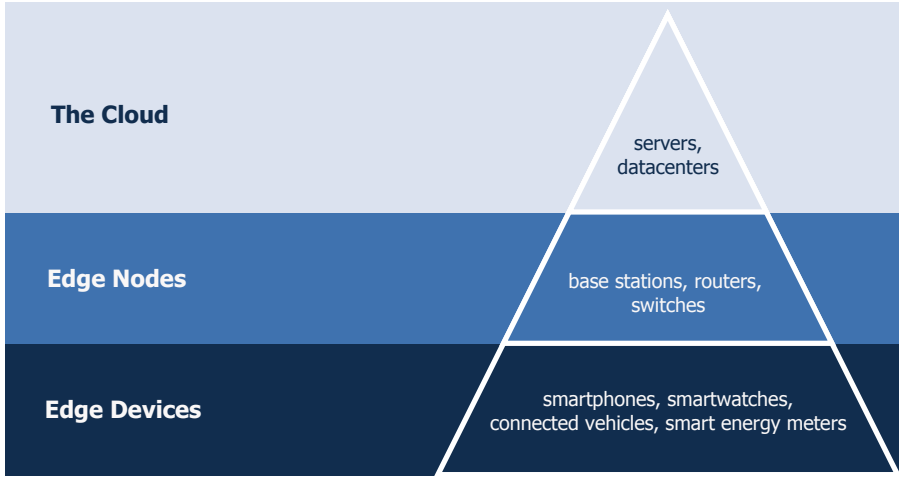

Figure 4: The cloud and the edge, with example devices of each layer.

and transmission of relevant alerts only to users in the same geographical area [52–54]. Furthermore, edge computing allows the design of applications that transfer only filtered or aggregated data to the cloud, offering much stronger privacy and security guarantees than pure cloud computing [50, 53, 55, 56].

4.3 Stream Processing at the Cloud and the Edge

Edge computing is a natural complement to stream processing since the former can facilitate the production of low latency results, one of the main goals of the latter. Thus, streaming queries can span the whole spectrum of the available processing devices to fully utilize the available hardware resources. In that scenario, queries can preprocess huge volumes of high-velocity raw data in edge devices before (optionally) transmitting it to the cloud for further processing [49]. That way, edge nodes can filter, aggregate, encrypt, encode, etc., the raw data received by end devices before they are transmitted to the cloud, removing uninteresting "noisy" measurements and increasing the privacy and security guarantees of the processing pipeline [11]. Edge computing can also assist at the opposite processing end, e.g., IoT sensors and similar end-devices with limited computational capacity, allowing them to offload some (or all) of the necessary computations to edge nodes [15].

Despite all the advantages of edge computing, expanding data processing from the cloud to the edge also comes with new challenges. As hinted above, in contrast to cloud infrastructure, edge nodes can have limited computational and storage resources and might lack the homogeneity of a data center. Thus, to take full advantage of the edge paradigm, it is necessary to develop processing systems designed around such heterogeneity and resource constraints. Such processing systems should be based on optimized algorithms and techniques that run equally well on high-powered servers and resource-constrained devices [15, 57], as well as programming paradigms that encourage parallel processing, allowing to split the work into multiple processors and/or nodes [58, 59]. Furthermore, queries deployed at the cloud and the edge can benefit from information that

identifies important, unusual, or problematic data points and assists in query debugging and compliance while also allowing to prioritize the transmission and storage of specific subsets of the data [60]. Such information can be extracted by data provenance, discussed below.

5 Data Provenance

Increasing data velocities together with applications and users that expect low-latency responses are leading to an increased number of analysts relying on data streaming for their queries. Such increased demand for data streaming means that non-experts might also be tasked with developing streaming queries. However, the transition from the widely understood relational data model to stream processing is not always trivial because, in contrast to batch processing, the *time* of the events is a *"first-class citizen"* in stream processing. For example, consider a simple database query that computes the average electricity consumption of each house in a Smart City application. If analysts want to express the above to a streaming query, they have to reason about the event- and processing-time, windows, delayed data, and out-of-order data [14]. This transition can aided by bringing the streaming model closer to the relational one by treating streams and tables as *"two sides of the same coin"* [61,62]. However, such approaches cannot completely hide the intrinsic differences of stream processing that, together with the inherent velocity of the inputs and outputs, can make it challenging to develop, test and debug streaming applications [63]. To make matters worse, depending on the application requirements and the resource constraints, it might not be possible or desirable to persistently store all input data, and thus it can be impractical to rerun a query on the exact same inputs in order, e.g., to verify that it is correct or to debug a problem [38].

Another issue is that, as streaming applications increase in complexity and importance, it is necessary to reason about their results and explain *unexpected behaviors*. Such robustness and predictability is crucial in use cases such as social sensors [30, 38] and Cyber-Physical Systems [24]. In the former case, large-scale social networks (e.g., Facebook) use data streaming to identify "trending" discussion topics in the network and verify the trustworthiness of such topics, e.g., by detecting misinformation campaigns by malicious actors. In the latter case, data streaming applications can be deployed in mission-critical settings such as avionics, health care, and road traffic to detect and react to anomalous events (e.g., a fire). In both cases, it is not only useful but frequently necessary to be able to explain 1) why an alert was produced, 2) all the alerts connected to a specific input, and 3) why an expected alert was not produced.

One technique for explaining the behavior of streaming applications is *provenance*, which is generally defined as *"any information that describes the production process of an end product"* [64]. Our work focuses on a specific subset of provenance known as *data provenance* (or simply *provenance* in the following), which tracks individual data items (i.e., tuples) through the application, along with the operators that process them. Data provenance can provide detailed explanations about the results of a streaming application, easing debugging and making the system more explainable. Below, we discuss

the aspects of provenance most related to the work in the thesis, referring the reader to [64] for a comprehensive study on provenance.

5.1 Backward and Forward Why-Provenance

A subset of data provenance called *why-provenance* is especially useful in Cyber-Physical Systems because it can explain important or critical events identified by queries and allow analysts to perform further investigation [64,65]. Why-provenance can either be backward, from the outputs to the query inputs, or forward, from the inputs to the query outputs [64].

Starting with *backward provenance*, we can consider a streaming application that produces alerts for blackouts in a Smart Grid by identifying households with zero power consumption in a specific time window. When an alert is produced, it could be important for the analyst to know which houses are without power so that necessary steps can be taken to resolve the issue. Fine-grained backward provenance can provide such information by connecting each output (e.g., alert) to all inputs that led to that output's generation (e.g., electricity consumption reports by smart meters in each house) [65].

Turning our attention to *forward provenance*, we have observed above that it might not be practical or desirable to store all the raw data generated by present-day data sources, either locally or by transmitting it to cloud storage. Instead, it might be advantageous to employ edge processing techniques to preprocess, filter, and aggregate the raw data and only keep "important" raw data points, e.g., inputs that led to an interesting output. Additionally, debugging, testing, or legal reasons might require analysts to be able to identify all outputs connected to a specific input, e.g., a privacy-sensitive data point [51]. Forward provenance can identify such "important" inputs by tracing each input to all outputs produced due to it, allowing applications to discard irrelevant inputs that are essentially "noise" [50] and identify outputs that were produced because of certain inputs.

Figure 5 shows two example queries (a) along with their backward (b) and forward (c) provenance. The two queries monitor the position and speed of vehicles: $Q1$ produces an alert every 5 minutes if a vehicle has been in a specific area R for more than 2/3 of the time in the last 15 minutes, and $Q2$ produces an alert every five minutes if the mean speed of a vehicle was higher than 110 km/h in the last 15 minutes. The left side of Figure 5a shows the movement of two cars, and the right side of the figure shows the alerts produced for these two cars over time. As seen in Figure 5b, the backward provenance of the two queries connects each alert α to the inputs t that contributed to its generation. Notice that the backward provenance graph contains duplicate information when an input contributes to multiple alerts. This issue is solved when using forward provenance, shown for the same alerts in Figure 5c. In that case, we can immediately traverse the graph to identify all outputs connected to a specific input without having to manually preprocess the graph. Furthermore, the forward provenance in this example is *live*: the ticks signify tuples that have *expired*, i.e., will not have any edges added to them in the future.

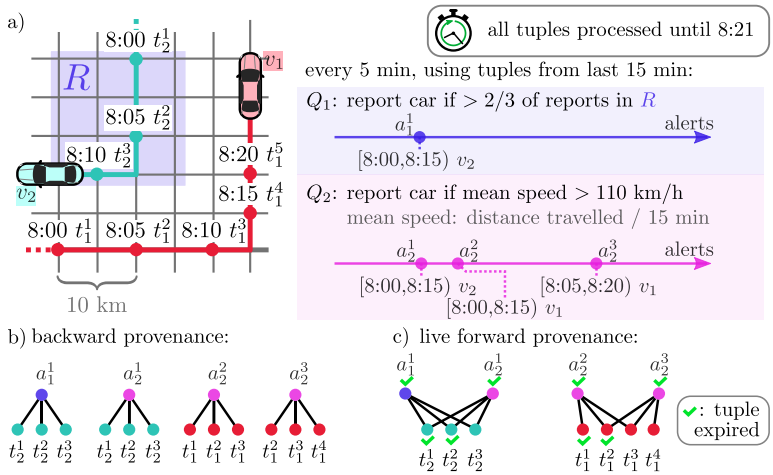

Figure 5: a) Two example queries that monitor the location ($Q1$) and the mean speed ($Q2$) of vehicles over time, with the right side of the figure showing the inputs over time; b) the backward provenance of the queries and the inputs shown in the figure; c) the live forward provenance of the queries for the same data. Figure originally presented in the paper introducing Ananke [60].

5.2 Why-Not Provenance

While backward and forward why-provenance can give fine-grained explanations about the relationships between critical inputs and outputs of streaming queries, they cannot help users that observe the *absence* of expected results from the query outputs. This task can be nonetheless achieved with *why-not provenance*, or *provenance of missing answers*, which can explain why an expected result was *not* produced by a query [64, 66]. Provenance of missing answers can be very beneficial in understanding the correctness of a query and the input data, allowing analysts to quickly identify issues without the need for manual, error-prone debugging [67], where the analyst would have to manually inspect the inputs and outputs of individual operators to understand the reasons for the absence of an expected result. Explanations for missing answers can have different forms, depending on which component can be responsible for the absence of the expected results [66]. More specifically, they can be 1) *instance-based*, describing changes to the input data that would cause the result to be produced, 2) *query-based*, describing which operators are responsible for pruning intermediate tuples that could have led to a result, or 3) *modification-based*, proposing changes to the query semantics that would produce the expected result. In this work, we study *query-based* explanations for streaming queries to identify tuples that could have led to a result but were discarded by some query operator, along with the responsible operator(s).

6 Scheduling

When data streaming applications are deployed on — possibly resource-constrained — edge nodes, the efficient utilization of all available computation

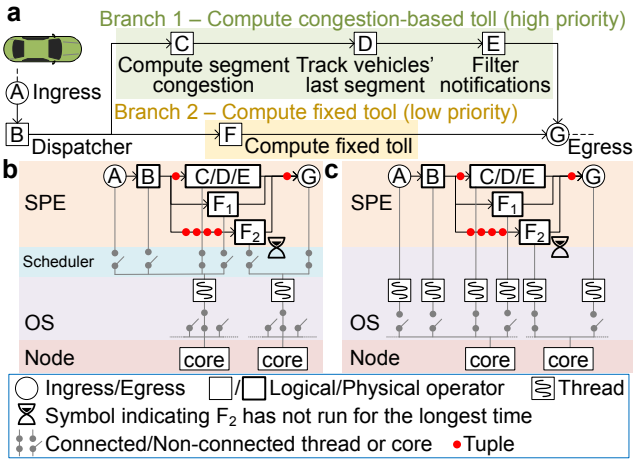

Figure 6: a) Logical query DAG of a simplified implementation of query from the Linear Road benchmark to compute tolls on highways; b) an example physical DAG of the query, along with an illustration of user-level scheduling; c) an example of OS-level scheduling for the same physical query DAG. The figure was first presented in the paper introducing Lachesis [68].

resources becomes a pressing issue [15, 24]. Furthermore, cloud infrastructure and edge nodes can have multi-tenancy, i.e., execute multiple applications simultaneously, each with its own performance goals and Quality-Of-Service (QoS) guarantees. Thus, it is also important for users to be able to control the resource allocation of the deployed streaming applications and optimize the heterogeneous goals of each of them. Such fine-grained resource allocation control is crucial in Cyber-Physical Systems, as interactions with the physical world are inherently sensitive on *timing*, especially when the system needs to respond to its environment [24, 29]. One way to achieve such fine-grained control and efficiency is through custom scheduling.

Scheduling is defined as the process of assigning certain units of work to specific resources. In data streaming, scheduling is used to refer both to *operator placement* and *thread scheduling*. The former, outside the scope of this thesis, refers to choosing where to deploy (e.g., which node) each processing unit (e.g., operator or query) in order to balance the processing load and minimize communication overheads and improve performance metrics such as throughput and latency [69, 70]. Thread scheduling, on the other hand, refers to prioritizing specific computations inside a processing group [71] and is usually more fine-grained than operator placement in the frequency of decisions as well as in the magnitude of the scheduled units. Notice that operator placement and thread scheduling are orthogonal to each other and can be used in conjunction to combine their benefits.

Custom operator scheduling in SPEs is usually done at the *user-level*. This technique, already utilized by pioneer SPEs [12, 72], involves the SPE using only a few OS threads (usually close to the number of available CPU cores) with the scheduler running as a user-level component of the SPE and choosing which operator is executed by which Operating System (OS) thread, and for how long. User-level scheduling allows for fine-grained control of the scheduling

decisions, as the scheduler has access to detailed, low-level information about the runtime characteristics of the query and the SPE. An alternative to user-level scheduling is OS-level scheduling, an approach frequently adopted by modern SPEs [37, 73]. In OS-level scheduling, the SPE does not include a separate scheduler component; instead, each physical operator is assigned its own OS thread, which in turn is scheduled by the OS scheduler.

Figure 6 outlines user- and OS-level scheduling on an example query. More specifically, Figure 6a shows a simplified implementation of a query from the Linear Road benchmark [74] that computes, for a set of highways, variable tolls based on congestion (Branch 1) and fixed tolls (Branch 2). Figures 6b/c show a possible transformation of the logical DAG of the query to a physical DAG, where (logical) operators C, D, E are fused together into the same physical operator and operator F is replicated twice. In Figure 6b, the SPE uses only two OS threads to run the operators. The operators are scheduled by a user-level scheduler that controls which operator runs on which thread and for how long (top switches in the figure). The two OS threads of the SPE are eventually scheduled by the OS (bottom switches in the figure). On the other hand, in Figure 6c, each operator is assigned to a dedicated OS-level thread, and the Operating System scheduler is solely responsible for the scheduling decisions. OS- and user-level scheduling have their own advantages and disadvantages, which are outlined later in this chapter.

7 Research Problems and State-of-the-Art

7.1 Data Provenance

As discussed in §5, data provenance can be valuable in explaining the behavior of streaming queries, allowing analysts to understand the presence or absence of specific outputs, making debugging easier, and helping validate the logic of streaming queries. Though data provenance has been studied in detail in the context of databases [64, 75, 76], it has not been a focus of stream processing research and practice until recently. Initial approaches for streaming provenance were coarse-grained [77, 78], tracing time intervals instead of individual tuples, or producing only approximate answers [79]. Other approaches focused on specialized use cases such as query debugging and data visualization [63, 80], while they also required manual guidance by the user.

Backward Provenance In order to track fine-grained backward provenance in data streaming queries, it is necessary to link each output to the inputs that led to its generation and vice-versa, something that previous works have shown to be an intrinsically heavy operation [65], with the potential to severely limit the performance of the original streaming query, especially when the data volume and velocity is high. The previous state-of-the-art solution for streaming backward why-provenance, called Ariadne, is based on instrumenting (i.e., encapsulating) the operators of streaming queries so that each processed tuple is annotated with provenance-specific metadata, allowing to link each output to the inputs that led to its generation in a fine-grained manner [65, 81]. However, this technique can lead to annotations that grow arbitrarily large,

increasing the memory impact of the provenance while it also requires the system to (temporarily) maintain *all* the raw input data, causing potentially prohibitive overheads, especially when streaming applications are deployed in resource-constrained edge devices [15, 57, 78]. These limitations lead us to our first research question about the development of efficient ways to record fine-grained backward provenance in stream processing:

Research Question 1

How can we maintain fine-grained backward data provenance in streaming systems with small processing and memory overheads?

Live Forward Provenance Forward provenance in streaming applications could be produced by traversing the backward provenance graph in reverse, but such an approach is inefficient in the streaming context. More specifically, the above approach would require maintaining the raw backward provenance in order to answer forward provenance questions. Since each input can lead to multiple outputs, such storage could maintain potentially huge volumes of duplicate information and cause significant overheads in resource-constrained devices. A general-purpose approach should deduplicate the backward provenance, which in turn requires the analyst to find a point in (event) time after which a source tuple will not contribute to any more results. This problem is not explored by previous works, which only focus on specific debugging and visualization use cases [77, 80]. Other previous works hint at the use of static query analysis to identify tuples that cannot contribute to any more results but do not study the problem in detail [65]. The following question expresses this gap in the research on live forward provenance for stream processing:

Research Question 2

How can we enrich data streaming frameworks that deliver backward provenance to efficiently provide live, duplicate-free, fine-grained, forward provenance for arbitrarily complex sets of queries?

Why-Not Provenance Provenance of missing answers has been studied in the context of relational queries, (reverse) top-k [82, 83], and skyline queries [84, 85], as well as in the context of more general architectures [86]. When focusing on query-based explanations, "Why Not?" [67] discusses the problem of missing answers in workflows when the analyst cannot inspect the input dataset and/or alter the query, proposing a solution that relies on replaying the query (with the same inputs) to identify the transformations that discarded the latest successors of tuples that could have contributed to the expected answer. NedExplain [66] follows a similar technique that, however, uses more accurate identification of source data to return more comprehensive and correct answers. Other works such as [87] compute query-based explanations in queries with nested data through reparameterizations of the operators, which allows them to include additional operators, such as projections, in the possible explanations.

However, the above batch-focused solutions are designed with the assumption that the query can be replayed with identical input data, which is usually

not the case in data streaming queries. This is because streaming queries focus on a one-pass analysis of input data that might be inefficient to maintain persistently [13]. Thus, replaying a streaming query with the same data to explain missing answers goes against the foundational principles of stream processing. Furthermore, the temporal nature of stream processing introduces the possibility of results that are not really missing but just delayed, something not present in traditional databases, which are based on a closed-world assumption. In particular, since streaming queries continuously process data, explanations for missing answers in stream processing must treat time as a "first-class citizen". The above challenges necessitate new research on missing answers, specifically in the context of general-purpose stream processing. This area has seen minimal research, apart from a single approach by Song et al. [88], which focuses solely on Complex Event Processing (CEP) and returns explanations that focus on time and not on all the attributes of the tuples. Thus, in order to allow analysts that rely on stream processing to get the benefits offered by why-not provenance, we want to answer the following research question:

Research Question 3

How can we efficiently monitor user-defined expectations about query results and explain missing answers in a streaming manner?

7.2 Custom Scheduling

Previous works have demonstrated improvements in the Quality-of-Service of streaming queries and the utilization of the available resources through the enforcement of custom scheduling policies [12, 69, 70, 89] and have proposed specific algorithms that optimize user-defined performance metrics such as latency, throughput and fairness [12, 72, 90–93].

User-Level Scheduling Pioneer SPEs, such as Aurora, included their own user-level schedulers, which were focused on running many operators on hardware with a small number of CPU cores, with the goal of optimizing throughput, latency, memory, or other Quality-of-Service metrics [94, 95]. As discussed above, modern SPEs, such as Apache Flink and Apache Storm, rely on OS-level scheduling, where each physical operator is assigned to a separate OS thread whose scheduling is controlled by the OS scheduler. While the OS scheduler is sophisticated and comprehensively tested on a wide range of workloads, it is usually unaware of the particular goals of data streaming applications. Thus, a research problem is how to easily allow practitioners and researchers to implement custom scheduling policies in modern SPEs, at a high level of abstraction without needing to re-implement the scheduler for each new SPE. This challenge is captured by the following research question:

Research Question 4

How can we provide resource-efficient and application-aware thread scheduling for streaming systems?

OS-Level Scheduling State-of-the-art scheduling solutions for stream processing require alterations to the core runtime of the SPE in order to schedule the query operators as user-level threads [43, 96]. While these alterations allow for fine-grained control of the scheduling decisions, the tight coupling between the implementation of the SPE and the scheduler can come with huge implementation and compatibility risks. Furthermore, user-level scheduling foregoes valuable facilities provided by the scheduler of the OS, such as the transparent handling of blocking operations and the transparent scheduling of operators belonging to different processes (of the same or different SPEs). Thus, it might be beneficial for SPE users to have access to custom scheduling without changing the SPE. Attempts in that direction have been made in Apache Storm, which can use Linux mechanisms such as `nice` and `cgroup` [97] to control resource allocation in a coarse-grained manner, as well as in resource controllers [98, 99], that aim to reduce Quality-of-Service violations by controlling how many resources are given to each application in multi-tenancy scenarios. However, previous works have not explored fine-grained schedulers that can control individual streaming operators through the use of OS mechanisms, as expressed by the research question below:

Research Question 5

Is it beneficial to implement custom scheduling by assisting the OS instead of altering the SPE to rely on a user-level scheduler?

7.3 Full Utilization of Edge and Cloud Infrastructure Through Parallel and Distributed Computation

Increasingly challenging analyses and deployments at resource-constrained edge nodes make data parallelization a necessity for streaming applications, especially when the latter require high Quality-of-Service [45, 100–102]. As discussed in §3, SPEs can transparently parallelize and distribute their native operators with minimal user involvement. However, stream processing extensions such as provenance might require the usage of custom logic not already included in such native operators. Thus our contributions should be designed from the ground up with scalability in mind so that they can keep up with the increasing requirements of modern stream processing applications. Furthermore, custom scheduling solutions that control the resource allocation of SPEs must be able to handle parallel and distributed deployments with minimal user interaction so that they can be used in conjunction with real-world query deployments. Such issues are described by our final research question:

Research Question 6

How can we allow the provenance and scheduling extensions for streaming queries to take advantage of modern parallel and distributed architectures to run efficiently in both low- and higher-end devices?

Table 1: Research questions defined in §7 and the thesis chapters that address them, together with the name of the technique discussed in each chapter.

	Part II: Provenance			**Part III: Scheduling**	
	Chapter A *GeneaLog*	**Chapter B** *Ananke*	**Chapter C** *Erebus*	**Chapter D** *Haren*	**Chapter E** *Lachesis*
RQ1	●	●	●	○	○
RQ2	○	●	●	○	○
RQ3	○	○	●	○	○
RQ4	○	○	○	●	●
RQ5	○	○	○	○	●
RQ6	●	●	●	●	●

8 Thesis Contributions

In this part, we briefly describe the main contributions of each chapter of the thesis concerning the research questions defined in §7. Table 1 outlines the chapters of the thesis, along with the name of the technique discussed in each chapter and the research question addressed by it.

8.1 Explainable Stream Processing Through Provenance

Low-Overhead Backward Provenance

We begin discussing data provenance in Chapter A by presenting our state-of-the-art solution for fine-grained backward data provenance in data streaming, called *GeneaLog*. GeneaLog is a framework that improves on the previous state-of-the-art by minimizing the overheads of backward provenance in stream processing, both in single- and multi-node deployments. It achieves this by using a small set of fixed-size metadata for each query tuple, in contrast with the variable-length tuple metadata annotations used in the previous state-of-the-art. Furthermore, GeneaLog utilizes process-level memory management facilities to avoid maintaining all input data and instead distinguishes on-the-fly the source tuples that actually contribute to some result. GeneaLog is designed to work in conjunction with parallelization and distribution of the operators, allowing the query to scale as necessary. GeneaLog is implemented and evaluated on top of two SPEs, Apache Flink [37] and Liebre [103]. Its evaluation using several real-world queries shows small overheads, even in resource-constrained devices, such as those deployed at the edge of Cyber-Physical Systems, in contrast to the previous state-of-the-art that resulted in at least one order of magnitude higher overheads and rapidly exhausted the memory of the evaluation devices.

Live Forward Provenance

In Chapter B, we continue our work on data provenance by proposing a framework, called *Ananke*, that efficiently records live forward provenance in data streaming queries. Ananke can use metadata from any backward provenance tool such as GeneaLog to produce a streaming bipartite graph of live forward provenance. The live forward provenance graph's vertices are

the deduplicated input tuples that contributed to some output as well as the outputs themselves. An input and an output vertex are connected in the graph if the input is part of the specific output's backward provenance. Furthermore, vertices contain liveness metadata which indicates if they have expired or if they can have more edges added to them in the future. We describe and formally prove the correctness of two implementations of Ananke, a compact one based on a custom operator and a more scalable one based on native SPE operators. We implement Ananke on top of Apache Flink, and we thoroughly evaluate it using real-world queries, comparing its overheads with the state-of-the-art in backward provenance, as well as ad-hoc, database-based solutions. As shown in the evaluation, Ananke's overheads are usually less than 5% compared to the state-of-the-art in backward provenance, indicating that Ananke can efficiently compute live forward provenance even in resource-constrained edge devices.

Result Expectations in Stream Processing

Chapter C focuses on why-not provenance, aiming to explain missing but expected results. It proposes *Erebus*, a framework that allows users to define their *expectations* about the results of a streaming query as boolean predicates and either validate that their expectations are met or produce query-based streaming explanations that describe why an expected result did not get produced by the query. In this chapter, we formally define the problem of missing answers in stream processing and study the challenges that arise from the temporal nature of the processing and are absent in previous works targeting traditional databases. We analytically solve these challenges, proving the solution's correctness, and use the developed techniques to implement Erebus as a framework on top of Apache Flink. We evaluate Erebus, the first framework of its kind, using real-world and synthetic queries deployed both in low- and higher-end devices. The evaluation shows that Erebus can produce explanations for the outputs of streaming queries while imposing only small to moderate overheads over the original streaming query. We consider the overheads of Erebus are an acceptable trade-off for its functionality, in light of the fact that explaining missing answers has been shown to cause overheads of more than one order of magnitude in related works.

8.2 Efficiency and Control Through Scheduling

User-Level Thread Scheduling for Data Streaming

Chapter D marks the beginning of the second part of the thesis that focuses on custom scheduling for stream processing applications. This chapter presents our framework called *Haren*, a user-level scheduler that can control the allocation of CPU resources to the operators of streaming queries to optimize user-defined performance goals. In contrast to previous works, Haren defines a general set of abstractions that can describe an SPE's runtime and the operator's behavior, providing APIs that allow users to develop custom scheduling policies to control resource allocation by executing operators as user-level threads that are *application-aware*. When applying a user-defined scheduling policy, Haren uses fine-grained synchronization mechanisms to efficiently compute the policy priorities and execute the selected operators with minimal scheduling

overheads. We design and implement Haren as a framework that can be integrated into an SPE to offer such custom scheduling with a small effort from the user, enabling the reuse of the scheduling policies and offering an optimized scheduling infrastructure across different SPEs. Haren's detailed evaluation shows that it can outperform the default OS scheduling in a variety of scenarios and achieve custom scheduling goals even in resource-constrained devices, where custom scheduling can be essential.

Customizing OS Scheduling of Streaming Queries

Chapter E introduces *Lachesis*, a scheduling middleware for stream processing that does not require any changes to the SPE or even redeployment of the running streaming queries. In contrast to state-of-the-art user-level schedulers like Haren, Lachesis runs as a standalone application and guides the scheduling of streaming operators through OS mechanisms such as `nice` and `cgroup`. This allows Lachesis to enforce custom, user-defined scheduling policies on streaming queries running on one or several nodes, and even on multiple SPEs. Lachesis is implemented in a modular fashion and is easily extensible to more SPEs, scheduling policies, and OS scheduling mechanisms. We extensively evaluate our implementation of Lachesis, using three different SPEs, and several real-world and synthetic workloads on local, distributed, and multi-SPE deployments. Our evaluation shows that not only can Lachesis bring significant performance improvements over the default OS scheduling, but it can also outperform previous state-of-the-art user-level schedulers while avoiding the implementation and maintainability risks that come with user-level scheduling.

9 Conclusions and Future Directions

This thesis proposes techniques that can improve the efficiency and usability of stream processing applications and prepare SPEs for new types of deployments that extend from the cloud and embrace new paradigms, such as the edge. We study data provenance and scheduling challenges in stream processing, proposing solutions that work equally well on high-powered servers and resource-constrained devices, such as those found at the edge of Cyber-Physical Systems. The work of this thesis is distilled into three state-of-the-art data provenance frameworks and two schedulers for streaming queries.

Regarding data provenance, *GeneaLog* dramatically improves the performance of backward streaming provenance compared to the previous state-of-the-art, *Ananke* is the first framework to provide live forward provenance for stream processing, and *Erebus* is the first framework to provide streaming explanations for the (missing) answers of streaming queries. Those three frameworks have been shown to induce small overheads, allowing provenance to be recorded together with the original streaming queries in a wide range of deployments while being able to scale to support increasing data volumes.

Regarding scheduling, *Haren* is our state-of-the-art user-level scheduler for SPEs with a focus on reusability, and *Lachesis* is a state-of-the-art middleware with a pioneering approach to controlling the scheduling of streaming queries through OS mechanisms. The evaluation of Haren and Lachesis shows that our scheduling frameworks can drastically improve the performance of streaming

queries compared to the default scheduling of modern SPEs while abstracting away the complexity of custom scheduling from the user.

In future work, it would be interesting to explore how to take advantage of the heterogeneous architectures present at the edge of Cyber-Physical Systems, using GPUs and other accelerators to improve the performance of stream processing applications. Furthermore, it is crucial to explore innovative techniques to reduce the impact of data transfers in modern SPEs, such as compression or smart serialization, which can potentially decrease the overheads of data-intensive operations, such as data provenance. Additionally, elasticity, i.e., the ability to automatically add or remove processing resources depending on the processing requirements, is an important research direction (especially at the edge) that needs further exploration. Machine learning techniques should also be explored both in the context of streaming data provenance (e.g., taking advantage of provenance for training purposes) and in the context of custom scheduling to explore the possibility of automated, goal-focused scheduling policies. Finally, location-aware processing is an exciting direction, along with better visualization techniques of stream processing that would further aid in understanding and debugging streaming pipelines.

Bibliography

[1] P. David, "The world's most valuable resource is no longer oil, but data," *The Economist*, vol. 6, May 2017. [Online]. Available: https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015. [Online]. Available: http://www.nature.com/articles/nature14539

[3] E. Al Nuaimi, H. Al Neyadi, N. Mohamed, and J. Al-Jaroodi, "Applications of big data to smart cities," *Journal of Internet Services and Applications*, vol. 6, no. 1, p. 25, Aug. 2015. [Online]. Available: http://www.jisajournal.com/content/6/1/25

[4] C. Lu and X. Tang, "Surpassing Human-Level Face Verification Performance on LFW with GaussianFace," in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015, p. 9.

[5] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. Ullah Khan, "The rise of "big data" on cloud computing: Review and open research issues," *Information Systems*, vol. 47, pp. 98–115, Jan. 2015. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0306437914001288

[6] J. Gantz and D. Reinsel, "Extracting Value from Chaos," *IDC IView*, vol. 1142, no. 2011, pp. 1–12, 2011.

[7] D. K. Hsiao and S. E. Madnick, "Database Machine Architecture in the Context of Information Technology Evolution," in *Proceedings of the Third International Conference on Very Large Data Bases - Volume 3*, ser. VLDB '77. VLDB Endowment, 1977, pp. 63–84. [Online]. Available: http://dl.acm.org/citation.cfm?id=1286580.1286586

[8] P. Gillin, "Will Teradata revive a market?" *Computer World*, pp. 43, 48, Feb. 1984. [Online]. Available: https://books.google.com/books?id=5pw6ePUC8YYC&pg=PA48

[9] Wikipedia contributors, "Wikipedia:Size of Wikipedia," *Wikipedia*, May 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Wikipedia:Size_of_Wikipedia&oldid=1086868944

[10] D. Reinsel, J. Gantz, and J. Rydning, "The Digitization of the World from Edge to Core," *Framingham: International Data Corporation*, vol. 16, p. 28, 2018.

[11] M. Chen, S. Mao, and Y. Liu, "Big Data: A Survey," *Mobile Networks and Applications*, vol. 19, no. 2, pp. 171–209, Apr. 2014. [Online]. Available: http://link.springer.com/10.1007/s11036-013-0489-0

[12] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: A new model and architecture for data stream management," *The VLDB Journal*, vol. 12, no. 2, pp. 120–139, Aug. 2003. [Online]. Available: https://doi.org/10.1007/s00778-003-0095-z

[13] M. Stonebraker, U. Çetintemel, and S. Zdonik, "The 8 requirements of real-time stream processing," *ACM Sigmod Record*, vol. 34, no. 4, pp. 42–47, 2005.

[14] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Fer Andez-Moctezuma, R. Lax, S. Mcveety, D. Mills, F. Perry, E. Schmidt, and S. W. Google, "The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing," *VLDB*, vol. 8, no. 12, pp. 1792–1803, 2015.

[15] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. S. Nikolopoulos, "Challenges and Opportunities in Edge Computing," in *2016 IEEE International Conference on Smart Cloud (SmartCloud)*, Nov. 2016, pp. 20–26.

[16] G.-H. Kim, S. Trimi, and J.-H. Chung, "Big-data applications in the government sector," *Communications of the ACM*, vol. 57, no. 3, pp. 78–85, Mar. 2014. [Online]. Available: https://dl.acm.org/doi/10.1145/2500873

[17] "More Than 500 Hours Of Content Are Now Being Uploaded To YouTube Every Minute," May 2019. [Online]. Available: https://www.tubefilter.com/2019/05/07/number-hours-video-uploaded-to-youtube-per-minute/

[18] "Facebook rules the social networking world with 1 billion users," Oct. 2012. [Online]. Available: https://www.pcworld.com/article/2011123/facebook-rules-the-social-networking-world-with-1-billion-users.html

[19] K. W. a. G. Wells, "Facebook's Timeline: 15 Years In," *Wall Street Journal*, Feb. 2019. [Online]. Available: https://www.wsj.com/articles/facebooks-timeline-15-years-in-11549276201

[20] R. Coppola and M. Morisio, "Connected car: Technologies, issues, future trends," *ACM Computing Surveys (CSUR)*, vol. 49, no. 3, p. 46, 2016.

[21] J. Gertner, "Behind GE's Vision For The Industrial Internet Of Things," Jun. 2014. [Online]. Available: https://www.fastcompany.com/3031272/can-jeff-immelt-really-make-the-world-1-better

[22] Q. Hardy, "Reasons to Believe the A.I. Boom Is Real," *The New York Times*, Jul. 2016. [Online]. Available: https://www.nytimes.com/2016/07/19/technology/reasons-to-believe-the-ai-boom-is-real.html

[23] R. Baheti and H. Gill, "Cyber-physical systems," *The impact of control technology*, vol. 12, no. 1, pp. 161–166, 2011.

[24] E. A. Lee, "Cyber Physical Systems: Design Challenges," in *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, May 2008, pp. 363–369.

[25] N. Jazdi, "Cyber physical systems in the context of Industry 4.0," in *2014 IEEE International Conference on Automation, Quality and Testing, Robotics*, May 2014, pp. 1–4.

[26] V. Gulisano, M. Almgren, and M. Papatriantafilou, "Online and scalable data validation in advanced metering infrastructures," in *IEEE PES Innovative Smart Grid Technologies, Europe*, Oct. 2014, pp. 1–6.

[27] ——, "METIS: A Two-Tier Intrusion Detection System for Advanced Metering Infrastructures," in *International Conference on Security and Privacy in Communication Networks*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, J. Tian, J. Jing, and M. Srivatsa, Eds. Springer International Publishing, 2015, pp. 51–68.

[28] J. van Rooij, J. Swetzén, V. Gulisano, M. Almgren, and M. Papatriantafilou, "eChIDNA: Continuous data validation in advanced metering infrastructures," in *2018 IEEE International Energy Conference (ENERGYCON)*, Jun. 2018, pp. 1–6.

[29] W. Wolf, "Cyber-physical systems," *Computer*, vol. 42, no. 03, pp. 88–89, 2009.

[30] T. Sakaki, M. Okazaki, and Y. Matsuo, "Earthquake Shakes Twitter Users: Real-Time Event Detection by Social Sensors," in *Proceedings of the 19th International Conference on World Wide Web*, ser. WWW '10. New York, NY, USA: ACM, 2010, pp. 851–860. [Online]. Available: http://doi.acm.org/10.1145/1772690.1772777

[31] S. Madden, "From Databases to Big Data," *IEEE Internet Computing*, vol. 16, no. 3, pp. 4–6, May 2012.

[32] L. E. Truesdell, *The Development of Punch Card Tabulation in the Bureau of the Census, 1890-1940: With Outlines of Actual Tabulation Programs*. US Government Printing Office, 1965.

[33] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1327452.1327492

[34] Apache, "Hadoop." [Online]. Available: https://hadoop.apache.org/

[35] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica, "Apache spark: A unified engine for big data processing," *Communications of The Acm*, vol. 59, no. 11, pp. 56–65, Oct. 2016. [Online]. Available: https://doi.org/10.1145/2934664

[36] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina *et al.*, "The design of the borealis stream processing engine." in *Second Biennial Conference on Innovative Data Systems Research (CIDR 2005)*, vol. 5, Asilomar, CA, USA, Jan. 2005, pp. 277–289.

[37] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache Flink: Stream and batch processing in a single engine," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 36, no. 4, pp. 28–38, 2015.

[38] G. J. Chen, J. L. Wiener, S. Iyer, A. Jaiswal, R. L. N. Simha, W. Wang, K. Wilfong, T. Williamson, and S. Yilmaz, "Realtime data processing at Facebook," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, vol. 26-June-2016, 2016, pp. 1087–1098.

[39] B. Chandramouli, J. Goldstein, M. Barnett, R. Deline, D. Fisher, J. C. Platt, J. F. Terwilliger, and J. Wernsing, "Trill : A High-Performance Incremental Query Processor for Diverse Analytics," *VLDB - Very Large Data Bases*, vol. 8, no. 4, pp. 401–412, 2015.

[40] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. M. Patel, K. Ramasamy, and S. Taneja, "Twitter heron: Stream processing at scale," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '15. New York, NY, USA: ACM, 2015, pp. 239–250. [Online]. Available: http://doi.acm.org/10.1145/2723372.2742788

[41] S. A. Noghabi, K. Paramasivam, Y. Pan, N. Ramesh, J. Bringhurst, I. Gupta, and R. H. Campbell, "Samza: Stateful Scalable Stream Processing at LinkedIn," *Proceedings of the VLDB Endowment*, pp. 1634–1645, 2017. [Online]. Available: https://doi.org/10.14778/3137765.3137770

[42] V. Gulisano, "StreamCloud: An elastic parallel-distributed stream processing engine," Ph.D. dissertation, Universidad Politécnica de Madrid, 2012.

[43] X. Fu, T. Ghaffar, J. C. Davis, and D. Lee, "Edgewise: A better stream processing engine for the edge," in *USENIX Annual Technical Conference (ATC) 19*. WA, USA: USENIX, Jul. 2019, pp. 929–946.

[44] M. Hirzel, R. Soulé, S. Schneider, B. Gedik, and R. Grimm, "A catalog of stream processing optimizations," *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, pp. 1–34, 2014.

[45] H. Röger and R. Mayer, "A Comprehensive Survey on Parallelization and Elasticity in Stream Processing," *ACM Computing Surveys*, vol. 52, no. 2, pp. 1–37, Apr. 2019. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3320149.3303849

[46] X. Chen, Y. Vigfusson, D. M. Blough, F. Zheng, K. Wu, and L. Hu, "GOVERNOR: Smoother Stream Processing Through Smarter Backpressure," in *2017 IEEE International Conference on Autonomic Computing (ICAC)*, Jul. 2017, pp. 145–154.

[47] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, Feb. 2009.

[48] S. Carson, A. Lundvall, R. Möller, S. Bävertoft, A. Jacobsson, G. Sellin, M. Björn, V. Singh, S. Carson, R. Ludwig, L. Wieweg, J. Edstam, P. Lindberg, and K. Öhman, "Ericsson Mobility Report," p. 32, Jun. 2016.

[49] S. Taherizadeh, A. C. Jones, I. Taylor, Z. Zhao, and V. Stankovski, "Monitoring self-adaptive applications within edge computing frameworks: A state-of-the-art review," *Journal of Systems and Software*, vol. 136, pp. 19–38, Feb. 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S016412121730256X

[50] I. Santos, M. Tilly, B. Chandramouli, and J. Goldstein, "DiAl: Distributed streaming analytics anywhere, anytime," *Proceedings of the VLDB Endowment*, vol. 6, no. 12, pp. 1386–1389, Aug. 2013. [Online]. Available: https://dl.acm.org/doi/10.14778/2536274.2536322

[51] E. Politou, E. Alepis, and C. Patsakis, "Forgetting personal data and revoking consent under the GDPR: Challenges and proposed solutions," *Journal of Cybersecurity*, vol. 4, no. 1, Jan. 2018. [Online]. Available: https://academic.oup.com/cybersecurity/article/doi/10.1093/cybsec/tyy001/4954056

[52] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog Computing: A Platform for Internet of Things and Analytics," in *Big Data and Internet of Things: A Roadmap for Smart Environments*, N. Bessis and C. Dobre, Eds. Cham: Springer International Publishing, 2014, vol. 546, pp. 169–186. [Online]. Available: http://link.springer.com/10.1007/978-3-319-05029-4_7

[53] M. Satyanarayanan, P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, W. Hu, and B. Amos, "Edge Analytics in the Internet of Things," *IEEE Pervasive Computing*, vol. 14, no. 2, pp. 24–31, Apr. 2015.

[54] T. H. Luan, L. Gao, Z. Li, Y. Xiang, G. Wei, and L. Sun, "Fog Computing: Focusing on Mobile Users at the Edge," *arXiv:1502.01815 [cs]*, Feb. 2015. [Online]. Available: http://arxiv.org/abs/1502.01815

[55] V. Tudor, V. Gulisano, M. Almgren, and M. Papatriantafilou, "BES: Differentially private event aggregation for large-scale IoT-Based systems," *Future Generation Computer Systems*, Jul. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X17322823

[56] 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. New York, NY, USA: Association for Computing Machinery, 2018, pp. 171–182. [Online]. Available: https://doi.org/10.1145/3210284.3210298

[57] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*. ACM, 2012, pp. 13–16.

[58] P. R. Geethakumari, V. Gulisano, B. J. Svensson, P. Trancoso, and I. Sourdis, "Single window stream aggregation using reconfigurable hardware," in *2017 International Conference on Field Programmable Technology (ICFPT)*. IEEE, 2017, pp. 112–119.

[59] Y. Nikolakopoulos, M. Papatriantafilou, P. Brauer, M. Lundqvist, V. Gulisano, and P. Tsigas, "Highly Concurrent Stream Synchronization in Many-Core Embedded Systems," in *Proceedings of the Third ACM International Workshop on Many-Core Embedded Systems - MES '16*. Seoul, Republic of Korea: ACM Press, 2016, pp. 2–9. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2934495.2934496

[60] D. Palyvos-Giannas, B. Havers, M. Papatriantafilou, and V. Gulisano, "Ananke: A streaming framework for live forward provenance," *Proceedings of the VLDB Endowment*, vol. 14, no. 3, pp. 391–403, 2020.

[61] M. J. Sax, G. Wang, M. Weidlich, and J.-C. Freytag, "Streams and Tables: Two Sides of the Same Coin," in *Proceedings of the International Workshop on Real-Time Business Intelligence and Analytics*. Rio de Janeiro Brazil: ACM, Aug. 2018, pp. 1–10. [Online]. Available: https://dl.acm.org/doi/10.1145/3242153.3242155

[62] E. Begoli, T. Akidau, F. Hueske, J. Hyde, K. Knight, and K. Knowles, "One SQL to Rule Them All," *Proceedings of the 2019 International Conference on Management of Data - SIGMOD '19*, pp. 1757–1772, 2019. [Online]. Available: http://arxiv.org/abs/1905.12133

[63] L. Battle, D. Fisher, R. DeLine, M. Barnett, B. Chandramouli, and J. Goldstein, "Making Sense of Temporal Queries with Interactive Visualization," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems - CHI '16*. Santa Clara, California, USA: ACM Press, 2016, pp. 5433–5443.

[64] M. Herschel, R. Diestelkämper, and H. Ben Lahmar, "A survey on provenance: What for? What form? What from?" *VLDB Journal*, vol. 26, no. 6, pp. 881–906, 2017.

[65] B. Glavic, K. S. Esmaili, P. M. Fischer, and N. Tatbul, "Efficient stream provenance via operator instrumentation," *ACM Transactions on Internet Technology*, vol. 14, no. 1, Aug. 2014. [Online]. Available: https://doi.org/10.1145/2633689

[66] N. Bidoit, M. Herschel, and K. Tzompanaki, "Query-Based Why-Not Provenance with NedExplain," in *Extending Database Technology (EDBT)*, 2014.

[67] A. Chapman and H. V. Jagadish, "Why not?" in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '09. New York, NY, USA: Association for Computing Machinery, Jun. 2009, pp. 523–534.

[68] D. Palyvos-Giannas, G. Mencagli, M. Papatriantafilou, and V. Gulisano, "Lachesis: A middleware for customizing OS scheduling of stream processing queries," in *Proceedings of the 22nd International Middleware Conference*, ser. Middleware '21. New York, NY, USA: Association for Computing Machinery, Dec. 2021, pp. 365–378. [Online]. Available: https://doi.org/10.1145/3464298.3493407

[69] L. Aniello, R. Baldoni, and L. Querzoni, "Adaptive online scheduling in storm," in *Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '13. New York, NY, USA: ACM, 2013, pp. 207–218. [Online]. Available: http://doi.acm.org/10.1145/2488222.2488267

[70] J. Xu, Z. Chen, J. Tang, and S. Su, "T-Storm: Traffic-Aware Online Scheduling in Storm," in *2014 IEEE 34th International Conference on Distributed Computing Systems*. USA: IEEE, Jun. 2014, pp. 535–544.

[71] M. A. Sharaf, P. K. Chrysanthis, A. Labrinidis, and K. Pruhs, "Efficient scheduling of heterogeneous continuous queries," in *Proceedings of the 32Nd International Conference on Very Large Data Bases*, ser. VLDB '06. Seoul, Korea: VLDB Endowment, 2006, pp. 511–522. [Online]. Available: http://dl.acm.org/citation.cfm?id=1182635.1164172

[72] B. Babcock, S. Babu, M. Datar, R. Motwani, and D. Thomas, "Operator scheduling in data stream systems," *The VLDB Journal*, vol. 13, no. 4, pp. 333–353, Dec. 2004. [Online]. Available: http://dx.doi.org/10.1007/s00778-004-0132-6

[73] Apache, "Storm," 2021. [Online]. Available: https://storm.apache.org/

[74] A. Arasu, M. Cherniack, E. Galvez, D. Maier, A. S. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibbetts, "Linear road: A stream data management benchmark," in *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, ser. VLDB '04. Toronto, Canada: VLDB Endowment, 2004, pp. 480–491. [Online]. Available: http://dl.acm.org/citation.cfm?id=1316689.1316732

[75] Y. Cui, J. Widom, and J. L. Wiener, "Tracing the lineage of view data in a warehousing environment," *ACM Transactions on Database Systems*, vol. 25, no. 2, pp. 179–227, Jun. 2000.

[76] J. Cheney, L. Chiticariu, and W.-C. Tan, "Provenance in databases: Why, how, and where," *Foundations and Trends in Databases*, vol. 1, no. 4, pp. 379–474, 2007. [Online]. Available: http://www.nowpublishers.com/article/Details/DBS-006

[77] N. N. Vijayakumar and B. Plale, "Towards Low Overhead Provenance Tracking in Near Real-Time Stream Filtering," in *Provenance and Annotation of Data*, L. Moreau and I. Foster, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 46–54.

[78] M. Wang, M. Blount, J. Davis, A. Misra, and D. Sow, "A time-and-value centric provenance model and architecture for medical event streams," in *Proceedings of the 1st ACM SIGMOBILE International Workshop on Systems and Networking Support for Healthcare and Assisted Living Environments*, ser. HealthNet '07. New York, NY, USA: ACM, 2007, pp. 95–100. [Online]. Available: http://doi.acm.org/10.1145/1248054.1248082

[79] M. R. Huq, A. Wombacher, and P. M. Apers, "Adaptive inference of fine-grained data provenance to achieve high accuracy at lower storage costs," in *7th IEEE International Conference on E-Science, e-Science 2011*. USA: IEEE Computer Society, Dec. 2011, pp. 202–209.

[80] W. De Pauw, M. Leţia, B. Gedik, H. Andrade, A. Frenkiel, M. Pfeifer, and D. Sow, "Visual Debugging for Stream Processing Applications," in *Runtime Verification*, H. Barringer, Y. Falcone, B. Finkbeiner, K. Havelund, I. Lee, G. Pace, G. Roşu, O. Sokolsky, and N. Tillmann, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, vol. 6418, pp. 18–35.

[81] B. Glavic, K. Sheykh Esmaili, P. M. Fischer, and N. Tatbul, "Ariadne: Managing fine-grained provenance on data streams," in *Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '13. New York, NY, USA: Association for Computing Machinery, 2013, pp. 39–50. [Online]. Available: https://doi.org/10.1145/2488222.2488256

[82] Z. He and E. Lo, "Answering why-not questions on top-k queries," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 6, pp. 1300–1315, 2014.

[83] Y. Gao, Q. Liu, G. Chen, B. Zheng, and L. Zhou, "Answering Why-Not Questions on Reverse Top-k Queries," *Proceedings of the VLDB Endowment: 41st VLDB 2015, August 31 - September 4, Kohala Coast, Hawaii*, vol. 8, no. 7, pp. 738–749, Sep. 2015.

[84] M. S. Islam, R. Zhou, and C. Liu, "On answering why-not questions in reverse skyline queries," in *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, 2013, pp. 973–984.

[85] S. Chester and I. Assent, "Explanations for skyline query results," in *EDBT*, 2015, pp. 349–360.

[86] Y. Wu, M. Zhao, A. Haeberlen, W. Zhou, and B. T. Loo, "Diagnosing missing events in distributed systems with negative provenance," *ACM*

SIGCOMM Computer Communication Review, vol. 44, no. 4, pp. 383–394, Aug. 2014.

[87] R. Diestelkämper, S. Lee, M. Herschel, and B. Glavic, "To Not Miss the Forest for the Trees - A Holistic Approach for Explaining Missing Answers over Nested Data," in *Proceedings of the 2021 International Conference on Management of Data*, ser. SIGMOD/PODS '21. New York, NY, USA: Association for Computing Machinery, Jun. 2021, pp. 405–417.

[88] S. Song, R. Huang, Y. Gao, and J. Wang, "Why not match: On explanations of event pattern queries," in *Proceedings of the 2021 International Conference on Management of Data*, ser. SIGMOD/PODS '21. New York, NY, USA: Association for Computing Machinery, 2021, pp. 1705–1717.

[89] T. Li, Z. Xu, J. Tang, and Y. Wang, "Model-free control for distributed stream data processing using deep reinforcement learning," *Proc. VLDB Endow.*, vol. 11, no. 6, pp. 705–718, Feb. 2018. [Online]. Available: https://doi.org/10.14778/3199517.3199521

[90] B. Chandramouli, J. Goldstein, R. Barga, M. Riedewald, and I. Santos, "Accurate latency estimation in a distributed event processing system," *Proceedings - International Conference on Data Engineering*, pp. 255–266, 2011.

[91] B. Babcock, S. Babu, R. Motwani, and M. Datar, "Chain: Operator scheduling for memory minimization in data stream systems," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '03. New York, NY, USA: ACM, 2003, pp. 253–264. [Online]. Available: http://doi.acm.org/10.1145/872757.872789

[92] J. Wolf, N. Bansal, K. Hildrum, S. Parekh, D. Rajan, R. Wagle, K.-L. Wu, and L. Fleischer, "SODA: An optimizing scheduler for large-scale stream-based distributed computer systems," in *Middleware 2008*, V. Issarny and R. Schantz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 306–325.

[93] T. N. Pham, P. K. Chrysanthis, and A. Labrinidis, "Avoiding class warfare: Managing continuous queries with differentiated classes of service," *The VLDB Journal*, vol. 25, no. 2, pp. 197–221, Apr. 2016. [Online]. Available: http://dx.doi.org/10.1007/s00778-015-0411-4

[94] D. Carney, Ŭ. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik, "Monitoring Streams: A New Class of Data Management Applications," in *Proceedings of the 28th International Conference on Very Large Data Bases*, ser. VLDB '02. New York, NY, USA: VLDB Endowment, 2002, pp. 215–226.

[95] D. Carney, U. Çetintemel, A. Rasin, S. Zdonik, M. Cherniack, and M. Stonebraker, "Operator scheduling in a data stream manager," in *Proceedings of the 29th International Conference on Very Large Data*

Bases, ser. VLDB '03, vol. 29. Berlin, Germany: VLDB Endowment, 2003, pp. 838–849.

[96] D. Palyvos-Giannas, V. Gulisano, and M. Papatriantafilou, "Haren: A Framework for Ad-Hoc Thread Scheduling Policies for Data Streaming Applications," in *Proceedings of the 13th ACM International Conference on Distributed and Event-Based Systems*, ser. DEBS '19. Darmstadt, Germany: ACM, 2019, pp. 19–30.

[97] S. Hallyn, P. Zijlstra, J. Lelli, M. Kerrisk, C. Emde, T. Bjorkholm, M. Kuhn, and D. Wheeler, *Sched(7) Linux Programmers's Manual*, 5th ed., Aug. 2020.

[98] M. R. HoseinyFarahabady, J. Taheri, A. Y. Zomaya, and Z. Tari, "A dynamic resource controller for resolving quality of service issues in modern streaming processing engines," in *2020 IEEE 19th International Symposium on Network Computing and Applications (NCA)*. Cambridge, MA, USA: IEEE, 2020, pp. 1–8.

[99] M. R. HoseinyFarahabady, A. Jannesari, J. Taheri, W. Bao, A. Y. Zomaya, and Z. Tari, "Q-flink: A QoS-Aware controller for apache flink," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. Melbourne, VIC, Australia: IEEE, 2020, pp. 629–638.

[100] T. De Matteis and G. Mencagli, "Keep Calm and React with Foresight: Strategies for Low-Latency and Energy-Efficient Elastic Data Stream Processing," in *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*. ACM, 2016, pp. 13:1–13:12.

[101] ——, "Proactive Elasticity and Energy Awareness in Data Stream Processing," *Journal of Systems and Software*, vol. 127, no. C, pp. 302–319, 2017.

[102] N. Hidalgo, D. Wladdimiro, and E. Rosas, "Self-adaptive Processing Graph with Operator Fission for Elastic Stream Processing," *Journal of Systems and Software*, vol. 127, no. C, pp. 205–216, 2017.

[103] Open Source, "Liebre SPE," 2021. [Online]. Available: https://github.com/vincenzo-gulisano/Liebre

[104] M. H. Ali, C. Gerea, B. S. Raman, B. Sezgin, T. Tarnavski, T. Verona, P. Wang, P. Zabback, A. Ananthanarayan, A. Kirilov, M. Lu, A. Raizman, R. Krishnan, R. Schindlauer, T. Grabs, S. Bjeletich, B. Chandramouli, J. Goldstein, S. Bhat, Y. Li, V. Di Nicola, X. Wang, D. Maier, S. Grell, O. Nano, and I. Santos, "Microsoft CEP server and online behavioral targeting," *Proc. VLDB Endow.*, vol. 2, no. 2, pp. 1558–1561, Aug. 2009. [Online]. Available: https://doi.org/10.14778/1687553.1687590

[105] M. Balazinska, H. Balakrishnan, S. Madden, and M. Stonebraker, "Fault-tolerance in the borealis distributed stream processing system," in *Proceedings of the 2005 ACM SIGMOD International Conference on*

Management of Data, ser. SIGMOD '05. New York: ACM, 2005, pp. 13–24. [Online]. Available: http://doi.acm.org/10.1145/1066157.1066160

[106] E. Kalyvianaki, M. Fiscato, T. Salonidis, and P. Pietzuch, "THEMIS: Fairness in federated stream processing under overload," in *Proceedings of the 2016 International Conference on Management of Data*, ser. SIGMOD '16. New York, NY, USA: ACM, 2016, pp. 541–553. [Online]. Available: http://doi.acm.org/10.1145/2882903.2882943

[107] Y. Ji, H. Zhou, Z. Jerzak, A. Nica, G. Hackenbroich, and C. Fetzer, "Quality-driven continuous query execution over out-of-order data streams," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '15. New York: ACM, 2015, pp. 889–894. [Online]. Available: http://doi.acm.org/10.1145/2723372.2735371

[108] 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 Transactions on Parallel Computing*, vol. 4, no. 2, pp. 11:1–11:28, Oct. 2017.

[109] V. Gulisano, Y. Nikolakopoulos, M. Papatriantafilou, and P. Tsigas, "Scalejoin: A deterministic, disjoint-parallel and skew-resilient stream join," *IEEE Transactions on Big Data*, pp. 1–1, 2016.

[110] I. Walulya, D. Palyvos-Giannas, Y. Nikolakopoulos, V. Gulisano, M. Papatriantafilou, and P. Tsigas, "Viper: A module for communication-layer determinism and scaling in low-latency stream processing," *Future Generation Computer Systems*, vol. 88, pp. 297–308, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X17326791

[111] HardKernel, "Odroid-XU4," 2020. [Online]. Available: http://www.hardkernel.com

[112] P. Groth and L. Moreau, "PROV-Overview. An overview of the PROV family of documents," World Wide Web Consortium, Project Report, Apr. 2013. [Online]. Available: https://eprints.soton.ac.uk/356854/

[113] Y. R. Wang and S. E. Madnick, "A polygen model for heterogeneous database systems: The source tagging perspective," in *Proceedings of the 16th International Conference on Very Large Data Bases*, ser. VLDB '90. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990, pp. 519–538. [Online]. Available: http://dl.acm.org/citation.cfm?id=645916.758355

[114] J. Freire, D. Koop, E. Santos, and C. T. Silva, "Provenance for computational tasks: A survey," *Computing in Science and Engineering*, vol. 10, no. 3, pp. 11–21, 2008.

[115] S. B. Davidson and J. Freire, "Provenance and scientific workflows: Challenges and opportunities," in *Proceedings of the 2008 ACM SIGMOD*

International Conference on Management of Data, ser. SIGMOD '08. New York, NY, USA: ACM, 2008, pp. 1345–1350. [Online]. Available: http://doi.acm.org/10.1145/1376616.1376772

[116] A. Cuzzocrea, "Provenance research issues and challenges in the big data era," *Proceedings - International Computer Software and Applications Conference*, vol. 3, pp. 684–686, 2015.

[117] R. Mayer, B. Koldehofe, and K. Rothermel, "Predictable Low-Latency Event Detection With Parallel Complex Event Processing," *IEEE Internet of Things Journal*, vol. 2, no. 4, pp. 274–286, Aug. 2015.

[118] A. Lerner and D. E. Shasha, "The virtues and challenges of ad hoc + streams querying in finance," *IEEE Data Eng. Bull.*, vol. 26, no. 1, pp. 49–56, 2003. [Online]. Available: http://sites.computer.org/debull/A03mar/lerner.ps

[119] Y. Mei and S. Madden, "ZStream: A cost-based query processor for adaptively detecting composite events," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '09. New York, NY, USA: ACM, 2009, pp. 193–206. [Online]. Available: http://doi.acm.org/10.1145/1559845.1559867

[120] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman, "Efficient pattern matching over event streams," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '08. New York, NY, USA: ACM, 2008, pp. 147–160. [Online]. Available: http://doi.acm.org/10.1145/1376616.1376634

[121] T. Akidau, A. Balikov, K. Bekiroğlu, S. Chernyak, J. Haberman, R. Lax, S. McVeety, D. Mills, P. Nordstrom, and S. Whittle, "MillWheel: Fault-tolerant stream processing at internet scale," *Proceedings of the VLDB Endowment*, vol. 6, no. 11, pp. 1033–1044, 2013.

[122] J. . Hwang, M. Balazinska, A. Rasin, U. Cetintemel, M. Stonebraker, and S. Zdonik, "High-availability algorithms for distributed stream processing," in *21st International Conference on Data Engineering (ICDE'05)*. Tokyo, Japan: IEEE, Apr. 2005, pp. 779–790.

[123] B. Havers, R. Duvignau, H. Najdataei, V. Gulisano, M. Papatriantafilou, and A. C. Koppisetty, "DRIVEN: A framework for efficient data retrieval and clustering in vehicular networks," *Future Generation Computer Systems*, vol. 107, pp. 1–17, 2020.

[124] H. Najdataei, Y. Nikolakopoulos, V. Gulisano, and M. Papatriantafilou, "Continuous and Parallel LiDAR Point-Cloud Clustering," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. Vienna, Austria: IEEE Computer Society, Jul. 2018, pp. 671–684.

[125] R. Duvignau, V. Gulisano, M. Papatriantafilou, and V. Savic, "Streaming piecewise linear approximation for efficient data management in edge computing," in *Proceedings of the 34th ACM/SIGAPP Symposium on*

Applied Computing - SAC '19. Limassol, Cyprus: ACM Press, 2019, pp. 593–596. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3297280.3297552

[126] F. Pasqualetti, F. Dörfler, and F. Bullo, "Attack detection and identification in cyber-physical systems," *IEEE Transactions on Automatic Control*, vol. 58, no. 11, pp. 2715–2729, 2013.

[127] S. Salah, G. Maciá-Fernández, and J. E. Díaz-Verdejo, "A model-based survey of alert correlation techniques," *Computer Networks*, vol. 57, no. 5, pp. 1289–1317, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128612004124

[128] K. N. Dominiak and A. R. Kristensen, "Prioritizing alarms from sensor-based detection models in livestock production - A review on model performance and alarm reducing methods," *Computers and Electronics in Agriculture*, vol. 133, pp. 46–67, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168169916311620

[129] D. Palyvos-Giannas, V. Gulisano, and M. Papatriantafilou, "GeneaLog: Fine-grained data streaming provenance in cyber-physical systems," *Parallel Computing*, vol. 89, p. 102552, Nov. 2019.

[130] D. Crawl, J. Wang, and I. Altintas, "Provenance for MapReduce-Based data-intensive workflows," in *Proceedings of the 6th Workshop on Workflows in Support of Large-Scale Science*, ser. WORKS '11. New York, NY, USA: Association for Computing Machinery, 2011, pp. 21–30. [Online]. Available: https://doi.org/10.1145/2110497.2110501

[131] Open Source, "Ananke implementation," 2020. [Online]. Available: https://github.com/dmpalyvos/ananke

[132] Apache, "Beam," 2021. [Online]. Available: https://beam.apache.org/

[133] J.-H. Hwang, U. Cetintemel, and S. Zdonik, "Fast and Reliable Stream Processing over Wide Area Networks," in *2007 IEEE 23rd International Conference on Data Engineering Workshop*, Apr. 2007, pp. 604–613.

[134] J. Li, K. Tufte, V. Shkapenyuk, V. Papadimos, T. Johnson, and D. Maier, "Out-of-order processing: A new architecture for high-performance stream systems," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 274–288, 2008.

[135] Y. Zheng, X. Xie, and W.-Y. Ma, "Geolife: A collaborative social networking service among user, location and trajectory." *IEEE Data Eng. Bull.*, vol. 33, no. 2, pp. 32–39, 2010.

[136] M.-F. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan *et al.*, "Argoverse: 3D tracking and forecasting with rich maps," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 8740–8749.

[137] PostgreSQL, "PostgreSQL," 2020. [Online]. Available: https://www.postgresql.org

[138] SQLite, "SQLite," 2020. [Online]. Available: https://www.sqlite.org/

[139] MongoDB, "MongoDB," 2020. [Online]. Available: https://www.mongodb.com

[140] Neo4j, "Neo4j," 2020. [Online]. Available: https://neo4j.com/

[141] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen, and D. Wilkins, "A comparison of a graph database and a relational database: A data provenance perspective," in *Proceedings of the 48th Annual Southeast Regional Conference*, ser. ACM SE '10. New York, NY, USA: Association for Computing Machinery, 2010. [Online]. Available: https://doi.org/10.1145/1900008.1900067

[142] B. Gedik, H. Andrade, K.-L. Wu, P. S. Yu, and M. Doo, "SPADE: The system s declarative stream processing engine," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '08. New York, NY, USA: Association for Computing Machinery, Jun. 2008, pp. 1123–1134.

[143] Open Source, "Erebus implementation," 2022. [Online]. Available: https://github.com/dmpalyvos/erebus

[144] Z. Jerzak and H. Ziekow, "The DEBS 2014 grand challenge," in *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '14. New York, NY, USA: Association for Computing Machinery, May 2014, pp. 266–269.

[145] MovieLens, "MovieLens," 2022. [Online]. Available: https://www.kaggle.com/rounakbanik/the-movies-dataset

[146] Apache, "Kafka," 2022. [Online]. Available: https://kafka.apache.org/

[147] OpenJDK, "Java microbenchmark harness," 2021. [Online]. Available: https://github.com/openjdk/jmh

[148] N. Bidoit, M. Herschel, and A. Tzompanaki, "Efficient computation of polynomial explanations of why-not questions," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, ser. CIKM '15. New York, NY, USA: Association for Computing Machinery, 2015, pp. 713–722.

[149] N. Bidoit, M. Herschel, and K. Tzompanaki, "Immutably answering why-not questions for equivalent conjunctive queries," in *6th USENIX Workshop on the Theory and Practice of Provenance (TaPP 2014)*. Cologne: USENIX Association, Jun. 2014.

[150] R. Dechter, I. Meiri, and J. Pearl, "Temporal constraint networks," *Artificial Intelligence*, vol. 49, no. 1, pp. 61–95, May 1991.

[151] Y. Xing, S. Zdonik, and J. -. Hwang, "Dynamic load distribution in the Borealis stream processor," in *21st International Conference on Data Engineering (ICDE'05)*. Tokyo, Japan: IEEE, Apr. 2005, pp. 791–802.

[152] M. A. Sharaf, P. K. Chrysanthis, A. Labrinidis, and K. Pruhs, "Algorithms and metrics for processing multiple heterogeneous continuous queries," *ACM Transactions on Database Systems*, vol. 33, no. 1, pp. 5:1–5:44, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1331904.1331909

[153] M. A. Bender, S. Chakrabarti, and S. Muthukrishnan, "Flow and stretch metrics for scheduling continuous job streams," in *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '98. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1998, pp. 270–279. [Online]. Available: http://dl.acm.org/citation.cfm?id=314613.314715

[154] M. A. Sharaf, P. K. Chrysanthis, and A. Labrinidis, "Preemptive Rate-Based Operator Scheduling in a Data Stream Management System," in *Proceedings of the ACS/IEEE 2005 International Conference on Computer Systems and Applications*, ser. AICCSA '05. USA: IEEE Computer Society, 2005, pp. 46–I.

[155] D. Palyvos-Giannas, V. Gulisano, and M. Papatriantafilou, "GeneaLog: Fine-grained data streaming provenance at the edge," in *Proceedings of the 19th International Middleware Conference*, ser. Middleware '18. New York, NY, USA: ACM, 2018, pp. 227–238. [Online]. Available: http://doi.acm.org/10.1145/3274808.3274826

[156] T. Urhan and M. J. Franklin, "Dynamic pipeline scheduling for improving interactive query performance," in *Proceedings of the 27th International Conference on Very Large Data Bases*, ser. VLDB '01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 501–510. [Online]. Available: http://dl.acm.org/citation.cfm?id=645927.672188

[157] S. Acharya and S. Muthukrishnan, "Scheduling on-demand broadcasts: New metrics and algorithms," in *Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, ser. MobiCom '98. New York, NY, USA: ACM, 1998, pp. 43–54. [Online]. Available: http://doi.acm.org/10.1145/288235.288248

[158] S. Muthukrishnan, R. Rajaraman, A. Shaheen, and J. E. Gehrke, "Online scheduling to minimize average stretch," in *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, ser. FOCS '99. Washington, DC, USA: IEEE Computer Society, 1999, pp. 433–. [Online]. Available: http://dl.acm.org/citation.cfm?id=795665.796508

[159] L. A. Moakar, T. N. Pham, P. Neophytou, P. K. Chrysanthis, A. Labrinidis, and M. Sharaf, "Class-based continuous query scheduling for data streams," in *Proceedings of the Sixth International Workshop on Data Management for Sensor Networks*, ser. DMSN '09. New York, NY, USA: ACM, 2009, pp. 9:1–9:6. [Online]. Available: http://doi.acm.org/10.1145/1594187.1594199

[160] T. N. Pham, L. A. Moakar, P. K. Chrysanthis, and A. Labrinidis, "DILoS: A dynamic integrated load manager and scheduler for continuous queries," in *2011 IEEE 27th International Conference on Data Engineering Workshops*. USA: IEEE, Apr. 2011, pp. 10–15.

[161] H. Kagermann, "Change through Digitization—Value creation in the age of industry 4.0," in *Management of Permanent Change*, H. Albach, H. Meffert, A. Pinkwart, and R. Reichwald, Eds. Wiesbaden: Springer Fachmedien Wiesbaden, 2015, pp. 23–45.

[162] G. Theodorakis, A. Koliousis, P. R. Pietzuch, and H. Pirk, "LightSaber: Efficient window aggregation on multi-core processors," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '20. Portland, OR, USA: ACM, 2020.

[163] P. M. Grulich, B. Sebastian, S. Zeuch, J. Traub, J. von Bleichert, Z. Chen, T. Rabl, and V. Markl, "Grizzly: Efficient stream processing through adaptive query compilation," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 2487–2503. [Online]. Available: https://doi.org/10.1145/3318464.3389739

[164] M. D. de Assunção, A. da Silva Veith, and R. Buyya, "Distributed data stream processing and edge computing: A survey on resource elasticity and future directions," *Journal of Network and Computer Applications*, vol. 103, pp. 1–17, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804517303971

[165] A. S. Tanenbaum and H. Bos, *Modern Operating Systems*. Pearson, 2015.

[166] D. Palyvos-Giannas, V. Gulisano, and M. Papatriantafilou, "Haren: A middleware for ad-hoc thread scheduling policies in data streaming," in *Proceedings of the 20th International Middleware Conference Demos and Posters*, 2019, pp. 19–20.

[167] P. Garefalakis, K. Karanasos, and P. Pietzuch, "Neptune: Scheduling suspendable tasks for unified Stream/Batch applications," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SoCC '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 233–245. [Online]. Available: https://doi.org/10.1145/3357223.3362724

[168] P. Dobbelaere and K. S. Esmaili, "Kafka versus RabbitMQ: A comparative study of two industry reference Publish/Subscribe implementations: Industry paper," in *Proceedings of the 11th ACM International Conference on Distributed and Event-Based Systems*, ser. DEBS '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 227–238. [Online]. Available: https://doi.org/10.1145/3093742.3093908

[169] V. Cardellini, V. Grassi, F. L. Presti, and M. Nardelli, "On QoS-Aware scheduling of data stream applications over fog computing infrastructures," in *2015 IEEE Symposium on Computers and Communication (ISCC)*. Larnaca: IEEE, Jul. 2015, pp. 271–276.

[170] R. Love, *Linux Kernel Development*, 3rd ed. Boston, MA, United States: Addison-Wesley Professional, 2010.

[171] S. Hallyn and M. Kerrisk, *Cgroups(7) Linux Programmers's Manual*, 5th ed., Aug. 2020.

[172] A. Shukla, S. Chaturvedi, and Y. Simmhan, "RIoTBench: An IoT benchmark for distributed stream processing systems," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 21, p. e4257, 2017.

[173] D. L. Mills, M. J., B. J., and K. W. others, "RFC0958: Network time protocol (NTP)," RFC Editor / Internet Engineering Task Force (IETF), USA, Tech. Rep., 1985.

[174] J. C. Eidson, M. Fischer, and J. White, "IEEE-1588 Standard for a precision clock synchronization protocol for networked measurement and control systems," in *Proceedings of the 34th Annual Precise Time and Time Interval Systems and Applications Meeting*, 2002, pp. 243–254.

[175] Open Source, "Lachesis' implementation," 2021. [Online]. Available: https://github.com/dmpalyvos/lachesis

[176] ——, "Lachesis' evaluation artifacts," 2021. [Online]. Available: https://github.com/dmpalyvos/lachesis-evaluation

[177] C. Davis *et al.*, "Graphite," 2021. [Online]. Available: https://graphiteapp.org/

[178] S. Zhang, J. He, C. A. Zhou, and B. He, "BriskStream: Scaling stream processing on multicore architectures," in *Proceedings of the 2019 International Conference on Management of Data*, ser. SIGMOD '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 705–722.

[179] S. Zhang, Y. Wu, F. Zhang, and B. He, "Towards concurrent stateful stream processing on multicore processors," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. Dallas, TX, USA: IEEE, 2020, pp. 1537–1548.

[180] M. V. Bordin, D. Griebler, G. Mencagli, C. F. R. Geyer, and L. G. L. Fernandes, "DSPBench: A suite of benchmark applications for distributed data stream processing systems," *IEEE access : practical innovations, open solutions*, vol. 8, pp. 222 900–222 917, 2020.

[181] J. Li, N. K. Sharma, D. R. K. Ports, and S. D. Gribble, "Tales of the tail: Hardware, OS, and application-level sources of tail latency," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SOCC '14. New York, NY, USA: Association for Computing Machinery, 2014, pp. 1–14.

[182] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013.

[183] H. Hofmann, H. Wickham, and K. Kafadar, "Letter-value plots: Boxplots for large data," *Journal of Computational and Graphical Statistics*, vol. 26, no. 3, pp. 469–477, 2017.

[184] Docker, "Docker," 2021. [Online]. Available: https://www.docker.com/

[185] H. Miao, H. Park, M. Jeon, G. Pekhimenko, K. S. McKinley, and F. X. Lin, "StreamBox: Modern stream processing on a multicore machine," in *Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference*, ser. USENIX ATC '17. USA: USENIX Association, 2017, pp. 617–629.

[186] Linux Kernel, "Pressure stall information," 2021. [Online]. Available: https://www.kernel.org/doc/html/latest/accounting/psi.html