Research Summary: Deterministic, Explainable and Efficient Stream Processing

Research Summary: Deterministic, Explainable and Efficient Stream Processing

Dimitris Palyvos-Giannas
Chalmers University of Technology
Gothenburg, Sweden
palyvos@chalmers.se

Marina Papatriantafilou
Chalmers University of Technology
Gothenburg, Sweden
ptrianta@chalmers.se

Vincenzo Gulisano
Chalmers University of Technology
Gothenburg, Sweden
vincenzo.gulisano@chalmers.se

ABSTRACT

The vast amounts of data collected and processed by technologies such as Cyber-Physical Systems require new processing paradigms that can keep up with the increasing data volumes. *Edge computing* and *stream processing* are two such paradigms that, combined, allow users to process unbounded datasets in an online manner, delivering high-throughput, low-latency insights. Moving stream processing to the edge introduces challenges related to the heterogeneity and resource constraints of the processing infrastructure. In this work, we present state-of-the-art research results that improve the facilities of Stream Processing Engines (SPEs) with data provenance, custom scheduling, and other techniques that can support the usability and performance of streaming applications, spanning through the edge-cloud contexts, as needed.

CCS CONCEPTS

• **Information systems** → **Online analytical processing engines**; **Data provenance**; • **Software and its engineering** → **Scheduling**.

KEYWORDS

Stream Processing, Provenance, Scheduling, Determinism

ACM Reference Format:
Dimitris Palyvos-Giannas, Marina Papatriantafilou, and Vincenzo Gulisano. 2022. Research Summary: Deterministic, Explainable and Efficient Stream Processing. In *Proceedings of the 2022 Workshop on Advanced tools, programming languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems (ApPLIED '22), July 25, 2022, Salerno, Italy*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3524053.3542750

1 INTRODUCTION

The vast amounts of data we collect today have enabled novel applications such as advanced recommendation engines and image recognition algorithms [19]. These data-driven advances are fueled by the spread of technologies like social networks, the Internet of Things (IoT), and Cyber-Physical Systems (CPSs), which caused a dramatic *"increase in the volume of data that are difficult to store, process, and analyze through traditional database technologies"* [16]. Since data volumes (e.g., positional reports, environmental measurements, health data) are expected to increase, how can we help the data analysis infrastructures keep up?

Though *cloud computing* revolutionized data processing, it is not always feasible or desirable to transfer all the raw data to the cloud, i.e., due to bandwidth limitations or when data transfer introduces non-trivial delays [17, 30] (e.g., an autonomous vehicle can generate 72 terabytes per day [30]). Also, in many applications, only a tiny fraction of the raw data is important, so cloud processing can waste network bandwidth for a minimal increase in the extracted value [33, 36]. Lastly, privacy regulations might also prevent raw data from being uploaded to the cloud [29].

Edge computing (also known as fog computing [6]) aims to mitigate such issues by utilizing additional processing nodes between the user/sensor and the cloud, allowing applications to move (part of) the processing closer to the data sources, taking advantage of the increasing computational capacity of *edge devices* such as base stations, switches or routers [36]. By being closer to data sources, edge devices can reduce the response time and bandwidth usage of data processing by either processing all the data locally or performing initial filtering and aggregations of the data before sending it to the cloud for further processing [33]. In turn, this can result in taking timely decisions based on (processed) sensor data [26], *location awareness* [6, 20, 32, 35], and stronger privacy and security guarantees, compared to pure cloud computing [34, 35].

Challenges And Opportunities. To utilize the computational power of cloud environments, paradigms such as *stream processing* (or *data streaming*) have been introduced to *continuously* process unbounded datasets in an online manner, resulting in an ever-growing ecosystem of Stream Processing Engines (SPEs) [1–3, 8]. SPEs offer high-level programming models for the development of streaming queries and can transparently handle challenging tasks such as scaling and fault-tolerance, hiding such complexities from the user.

However, moving processing from the cloud to the edge comes with new challenges that are not handled by modern SPEs. From a model perspective, the distributed and parallel analysis offered by SPEs supports applications whose computational power comes from many devices in the cloud/edge spectrum. Nevertheless, in contrast to cloud devices, edge devices can have limited computational and storage resources and lack the homogeneity of a data center. Thus, to take full advantage of edge devices, it is necessary to develop processing systems designed around such heterogeneity and resource constraints. Such systems should be based on optimized algorithms and techniques that run equally well on high-powered servers and resource-constrained devices [36], as well as programming paradigms that encourage parallel processing, allowing to split the work into multiple processors and/or nodes [10, 22].

ApPLIED '22, July 25, 2022, Salerno, Italy

ACM ISBN 978-1-4503-9280-8/22/07…$15.00
https://doi.org/10.1145/3524053.3542750

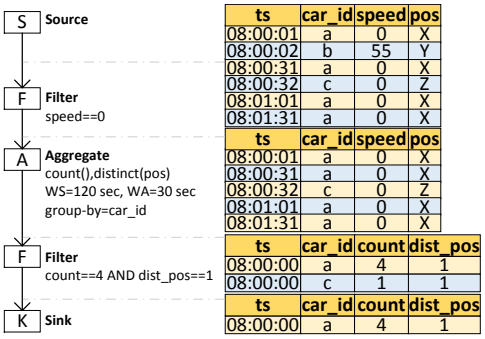

Figure 1: Sample streaming query identifying stopped vehicles in the Linear Road benchmark [4].

Contributions. In this research summary, we present state-of-the-art techniques that tackle some challenges related to broadening the processing to the edge too. More specifically, we discuss 1) how to efficiently trace/select important data to explain unexpected behaviors of streaming applications through *data provenance*, and 2) how to increase the efficiency of streaming applications and allow users to optimize for specific performance goals through custom *scheduling*. We also discuss other general techniques that can improve the usability and performance of streaming queries (e.g., deterministic processing and richer processing APIs).

Organization: we introduce background terms in § 2, we discuss data provenance contributions in § 3, scheduling contributions in § 4, and other general contributions in § 5, before concluding in § 6.

2 STREAM PROCESSING BACKGROUND

Our work builds on the DataFlow model [2] that is adopted by SPEs such as Apache Flink [8]. *Streams* are unbounded sequences of *tuples*, each with a *timestamp* (i.e., its *event-time*, 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 generate *source tuples* corresponding 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 with user-defined functions and can discard and/or forward (potentially new) tuples downstream in the query. Query results eventually reach the Sinks 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 are similar to their relational counterparts, and also allow defining *custom operators*. Operators are either *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 the windows. SPEs ensure correctness even in parallel/distributed executions with potentially out-of-order input data through mechanisms such as *watermarks* [2], that provide guarantees on the degree of out-of-orderness present in each stream, or *sorting* the input tuples of each operator, based on their timestamps [13].

Figure 1 shows an example query (originally presented in [23]) based on the Linear Road benchmark [4], which simulates vehicular traffic on linear expressways. Each vehicle sends a position report every 30 seconds, each of which is processed to identify stopped vehicles, that is, vehicles whose latest four reports had zero speed and the same position. The Aggregate's window size (WS) and window advance (WA) are set to 120 and 30 seconds, meaning that the Aggregate produces a result every 30 seconds (for each vehicle), based on the data of the last 120 seconds.

3 SMART DATA TRACING AND SELECTION WITH DATA PROVENANCE

Data provenance refers to the process of tracing individual data items (i.e., tuples) through an application, along with the operators that process them [18], in order to provide detailed explanations about the results of a streaming application, easing debugging and making the system more explainable. Data provenance can prove especially useful in Cyber-Physical Systems by explaining outputs corresponding to critical events and allowing further investigation [12]. Additionally, as discussed in § 1, in many scenarios, it might not be practical or desirable to store all the raw data, 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 sensitive raw data points, e.g., inputs that led to an interesting output. Thus, for debugging, testing, or legal reasons [29] analysts might want to identify all outputs connected to a specific input, e.g., a privacy-sensitive data point. Provenance can help users identify such important inputs and/or outputs, allowing applications to discard irrelevant data that is essentially "noise" [31].

Efficient Streaming Backward Provenance. To track provenance in streaming queries, each query output must be linked with all inputs that led to its generation. This is an intrinsically heavy operation, the overhead of which can be prohibitive when streaming queries are deployed in resource-constrained edge devices [6, 12, 36].

In GeneaLog [23, 24],we present a state-of-the-art solution for fine-grained backward data provenance in data streaming that improves on previous works by significantly lowering the provenance overheads. GeneaLog uses constant-size tuple annotations and process-level memory management facilities to avoid maintaining all input data. Instead, it distinguishes on-the-fly all source tuples

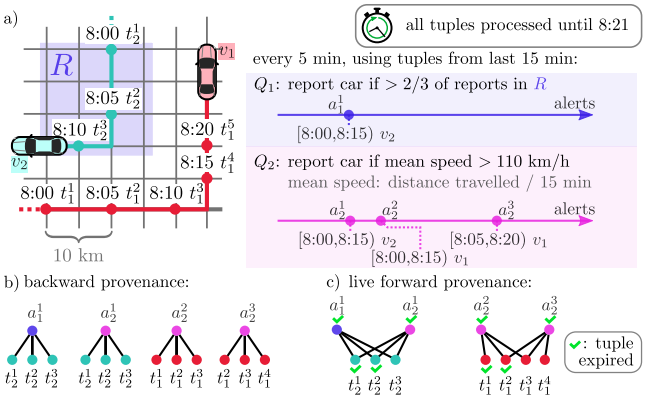

Figure 2: a) Two sample queries to monitor car location and mean speed (all tuples up to time 8:21 are processed), and their b) backward and c) live forward provenance graphs.

that actually contribute to some result. Additionally, GeneaLog is designed to work in parallel and distributed deployments, allowing the query to scale with the processing load. Figure 2a (originally presented in [26]) shows two example streaming queries monitoring a vehicular network to identify cars that are in a specific area (Q_1) or are speeding (Q_2). The backward provenance (GeneaLog's output) is shown in Figure 2b. Notice that, in this provenance graph, input tuples that contribute to multiple results are repeated, an issue that is handled by live forward provenance, discussed below. As evaluated using real-world queries and two SPEs, GeneaLog results in a small overhead, even in resource-constrained devices, showing performance improvements of more than one order of magnitude compared to the previous state-of-the-art.

Live Forward Provenance. Backward provenance provides valuable insights to e.g., debug applications. Since it focuses on tracing how each individual result is built, such insights are nonetheless of little help in answering questions such as *"which output depends on raw data that could contribute to new results?"* or *"which input data contributed to at least 5 results?"* Forward provenance can answer such questions by tracing all results linked to specific inputs. Forward provenance can be produced by traversing the backward provenance graph (e.g., as it is produced by GeneaLog) in reverse, but that can include duplicates (when inputs contribute to multiple outputs), and it is not *live*, i.e., it cannot answer whether an input can contribute to more results in the future. A general live forward provenance solution requires deduplication of the backward provenance, which, in the context of stream processing, requires finding a point in time after which a source tuple will not contribute to any more results. This problem has not been explored by previous works, which, instead, focus on specialized solutions for debugging or visualization of subsets of the query's operators and data [18].

To address the above issues, we develop a framework, called *Ananke* [26], 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. Input vertices contain live metadata indicating if that input can contribute to more results in the future. An input and an output vertex are connected in the graph if the input is part of the specific output's backward provenance. An example of such a live forward provenance graph is shown in Figure 2c (for the queries of Figure 2a), with tuples being included in the graph exactly once and some of them marked with a green check-mark if the respective tuple has *expired* and will not have any vertices added to it in the future. We propose two implementations of Ananke that balance simplicity with scalability, and we prove their correctness.

Ananke's extensive evaluation, using Apache Flink and real-world use cases, shows that Ananke's overheads are small and close to the overheads of backward provenance. Figure 3, taken from the evaluation of Ananke, shows the performance overheads of GeneaLog and Ananke for a vehicular tracking query running on a resource-constrained device, which is similar to those deployed at the edge of Cyber-Physical Systems. As shown in the figure, both GeneaLog (GL) and the different implementations of Ananke

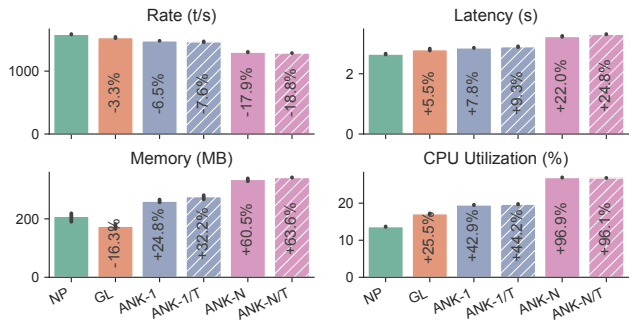

Figure 3: Performance overheads of backward (GeneaLog - GL) and live-forward provenance (Ananke - ANK) compared to the original query (NP) for a vehicular tracking query running on a resource-constrained edge-like device.

(ANK-) have low overheads compared to the original streaming query without provenance (NP): less than 8% impact in throughput and latency for the best performing implementations (GL, ANK-1).

4 CUSTOMIZING PERFORMANCE GOALS WITH THREAD SCHEDULING

When data streaming applications are deployed on edge nodes, it is crucial to efficiently utilize all available computational resources. Additionally, when cloud and edge nodes have *multi-tenancy* i.e., are concurrently executing applications with different performance goals and Quality-Of-Service requirements, it is important for users to be able to control the resource allocation of each application and balance their combined performance requirements. One way to achieve such fine-grained efficiency and control is through custom scheduling, i.e., by controlling how much CPU time is given to each streaming operator and the priority of the executions [27]. Scheduling decisions can either be controlled at the user-space by treating streaming operators as *user-level* threads or at the *OS-level*, where each operator is assigned to a dedicated Operating System (OS) thread and is scheduled by the OS scheduler (as is the default case in SPEs such as Apache Flink [8] and Apache Storm [3]).

Custom User-Level Scheduling. Our work in Haren [25] eases the transition of SPEs to custom user-level scheduling through a general scheduling framework that controls the allocation of CPU time to the operators with the goal of optimizing user-defined performance goals such as latency, throughput, and fairness [28]. Differently from previous works, Haren defines high-level abstractions that describe the runtime of an SPE and custom scheduling policies, hiding implementation details and allowing for code reuse. Haren controls resource allocation by running operators as user-level threads that are scheduled in an *application-aware* manner, following a custom user-defined scheduling policy. We design and implement Haren as a standalone framework that can be integrated into an SPE to offer such custom scheduling with minimal effort from the user compared to previous ad-hoc approaches. Haren's detailed evaluation shows that it can outperform the default OS scheduling in a variety of scenarios, achieving user-defined goals in resource-constrained devices where custom scheduling can be essential [28].

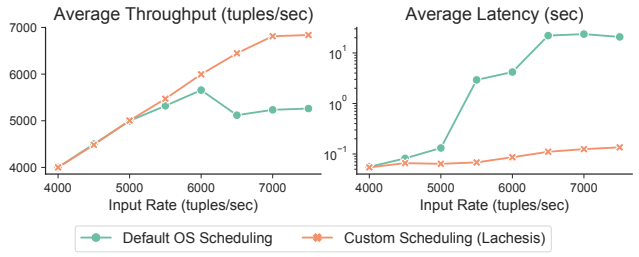

Figure 4: Performance benefits of custom scheduling with Lachesis for a streaming query from the Linear Road benchmark [4], for increasing input rates.

Custom Operating System Scheduling. State-of-the-art custom scheduling solutions for data streaming require alterations to the core runtime of the SPE [9] to be able to schedule the query operators as user-level threads. While these alterations allow for fine-grained control of the scheduling decisions, the tight coupling in the implementation of the SPE and the scheduler can introduce implementation and compatibility risks. Modern SPEs [3, 8] usually delegate the scheduling decisions to the underlying Operating System (OS). However, while OS schedulers are sophisticated and comprehensively tested on a wide range of workloads, they are not aware of the particular goals of data streaming applications.

Our work in Lachesis [27] explores an alternative approach to custom scheduling that does not require changes to the SPE. It proposes a standalone middleware that controls the scheduling decisions of the OS through mechanisms such as `nice` and `cgroup` to enforce custom scheduling policies in streaming queries running on one or several nodes and even on multiple SPEs. Lachesis is modular and easily extensible to support more SPEs, scheduling policies, and OS mechanisms. The extensive evaluation of Lachesis, with three SPEs, real-world and synthetic workloads, and different deployments, shows that it can bring significant performance improvements over the default OS scheduling and state-of-the-art user-level schedulers (up to +75% throughput and -1130x latency).

Figure 4, from Lachesis' evaluation, shows the average throughput and latency of a streaming query from the Linear Road benchmark [4] running on a resource-constrained device using Apache Storm [3], scheduled either using Lachesis or the default OS scheduling. As shown in the figure, as the input rate of the query increases, custom scheduling with Lachesis achieves significantly better average throughput and latency compared to the default OS scheduling.

5 OTHER TECHNIQUES TO SUPPORT CLOUD/EDGE STREAMING APPLICATIONS

Besides the support for edge to cloud analysis through provenance (see § 3) and custom scheduling (see § 4), other complementary approaches are worth mentioning.

One such approach is that of supporting efficient scaling of applications. Focusing on efficient parallel execution, for instance, one can notice that parallelization performs best when there is minimal synchronization and the parallel instances of the operators work independently as much as possible. The usual case is for such synchronization to be in place to support correct execution [5, 15, 38], ensuring results are produced by operators only when all the tuples contributing to a certain result have been processed. While SPEs commonly support correct and deterministic execution for order-insensitive analysis, they rely on users themselves to define deterministic order-sensitive analysis [21]. In this context, modules such as *Viper* [37] and *STRETCH* [21] introduce transparent methods to guarantee determinism in parallel stream processing (for both order-insensitive and order-sensitive analysis) without sacrificing efficiency and helping applications scale up before they scale out [11]. Both solutions rely on the *ScaleGate* [14] data structure which, in contrast with previous approaches, relies on lock-free synchronization to merge-sort the parallel streams.

A second aspect is to provide richer APIs than those of traditional SPEs. Besides the common native operators provided by most SPEs (§ 2), SPEs allow general-purpose operators whose semantics can be defined by the users. If, on the one hand, it is easier for users to express complex semantics through ad-hoc operators rather than a composition of native ones, this results, on the other hand, in two major downsides [26]. First, it requires users to define such ad-hoc operators so that their distributed and parallel execution can still be achieved with the same API offered by the underlying SPEs. Second, it hinders the portability of such ad-hoc operators across SPEs, as each SPE defines its own APIs for the custom operators it offers to users. As shown in [7], rich semantics like those offered by Spatio-Temporal Logics can be indeed enforced by automatically composing native operators, allowing users to express complex monitoring queries with compact notations and avoiding the manual and error-prone composition of large data pipelines.

6 CONCLUSIONS

We presented state-of-art techniques to address challenges that arise when deploying streaming queries at the edge. We presented two data provenance frameworks, *GeneaLog* and *Ananke*, that allow users to maintain backward and live forward provenance to identify important data items in stream processing while incurring only small overheads, even in resource-constrained devices. We also introduced two custom schedulers for SPEs: *Haren*, which allows users to control the scheduling decisions at the user level, and *Lachesis*, which guides the scheduling decisions through OS-level mechanisms. Both schedulers can improve the resource utilization of streaming queries and allow to optimize for custom performance goals. Lastly, we outlined other general solutions that improve the usability of stream processing by guaranteeing determinism and enriching the development APIs provided by modern SPEs.

ACKNOWLEDGMENTS

Work supported by the Swedish Government Agency for Innovation Systems VINNOVA, project "AutoSPADA" grant nr. DNR 2019-05884 in the funding program FFI: Strategic Vehicle Research and Innovation, the Swedish Foundation for Strategic Research, project "FiC" grant nr. GMT14-0032, the Swedish Research Council (Vetenskapsrådet) projects "HARE" grant nr. 2016-03800, EPITOME grant nr. 2021-05424, Chalmers Energy AoA framework projects INDEED, STAMINA, Production WP "Scalability, Big Data and AI".

REFERENCES

[1] Daniel J. Abadi, Don Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. 2003. Aurora: A New Model and Architecture for Data Stream Management. *The VLDB Journal* 12, 2 (Aug. 2003), 120–139. https://doi.org/10.1007/s00778-003-0095-z

[2] Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J Fer Andez-Moctezuma, Reuven Lax, Sam Mcveety, Daniel Mills, Frances Perry, Eric Schmidt, and Sam Whittle Google. 2015. The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing. *VLDB* 8, 12 (2015), 1792–1803. https://doi.org/10.14778/2824032.2824076

[3] Apache. 2021. Storm. Retrieved November 5, 2021 from https://storm.apache.org/

[4] Arvind Arasu, Mitch Cherniack, Eduardo Galvez, David Maier, Anurag S. Maskey, Esther Ryvkina, Michael Stonebraker, and Richard Tibbetts. 2004. Linear Road: A Stream Data Management Benchmark. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30 (VLDB '04)*. VLDB Endowment, Toronto, Canada, 480–491. http://dl.acm.org/citation.cfm?id=1316689.1316732

[5] Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, and Julian Shun. 2012. Internally Deterministic Parallel Algorithms Can Be Fast. In *Proceedings of the 17th SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*. ACM, 181–192.

[6] Flavio Bonomi, Rodolfo Milito, Preethi Natarajan, and Jiang Zhu. 2014. Fog Computing: A Platform for Internet of Things and Analytics. In *Big Data and Internet of Things: A Roadmap for Smart Environments*, Nik Bessis and Ciprian Dobre (Eds.). Vol. 546. Springer International Publishing, Cham, 169–186. https://doi.org/10.1007/978-3-319-05029-4_7

[7] Luca Bortolussi, Vincenzo Gulisano, Eric Medvet, and Dimitrios Palyvos-Giannas. 2019. Automatic Translation of Spatio-Temporal Logics to Streaming-Based Monitoring Applications for IoT-Equipped Autonomous Agents. In *Proceedings of the 6th International Workshop on Middleware and Applications for the Internet of Things*. 7–12.

[8] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache Flink: Stream and Batch Processing in a Single Engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 36, 4 (2015), 28–38.

[9] Xinwei Fu, Talha Ghaffar, James C Davis, and Dongyoon Lee. 2019. Edgewise: A Better Stream Processing Engine for the Edge. In *USENIX Annual Technical Conference (ATC) 19*. USENIX, WA, USA, 929–946.

[10] Prajith Ramakrishnan Geethakumari, Vincenzo Gulisano, Bo Joel Svensson, Pedro Trancoso, and Ioannis Sourdis. 2017. Single Window Stream Aggregation Using Reconfigurable Hardware. In *2017 International Conference on Field Programmable Technology (ICFPT)*. IEEE, 112–119.

[11] Phillip B. Gibbons. 2015. Big data: Scale down, scale up, scale out. In *IEEE International Parallel and Distributed Processing Symposium*. 3–3. https://doi.org/10.1109/IPDPS.2015.123

[12] Boris Glavic, Kyumars Sheykh Esmaili, Peter M. Fischer, and Nesime Tatbul. 2014. Efficient Stream Provenance via Operator Instrumentation. *ACM Transactions on Internet Technology* 14, 1, Article 7 (Aug. 2014). https://doi.org/10.1145/2633689

[13] Vincenzo Gulisano. 2012. *StreamCloud: An Elastic Parallel-Distributed Stream Processing Engine*. Ph. D. Dissertation. Universidad Politécnica de Madrid.

[14] Vincenzo Gulisano, Yiannis Nikolakopoulos, Marina Papatriantafilou, and Philippas Tsigas. 2016. Scalejoin: A Deterministic, Disjoint-Parallel and Skew-Resilient Stream Join. *IEEE Transactions on Big Data* (2016), 1–1. https://doi.org/10.1109/TBDATA.2016.2624274

[15] Vincenzo Gulisano, Alessandro V. Papadopoulos, Yiannis Nikolakopoulos, Marina Papatriantafilou, and Philippas Tsigas. 2017. Performance Modeling of Stream Joins. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems - DEBS '17*. ACM Press, Barcelona, Spain, 191–202. https://doi.org/10.1145/3093742.3093923

[16] Ibrahim Abaker Targio Hashem, Ibrar Yaqoob, Nor Badrul Anuar, Salimah Mokhtar, Abdullah Gani, and Samee Ullah Khan. 2015. The Rise of "Big Data" on Cloud Computing: Review and Open Research Issues. *Information Systems* 47 (Jan. 2015), 98–115. https://doi.org/10.1016/j.is.2014.07.006

[17] Bastian Havers, Romaric Duvignau, Hannaneh Najdataei, Vincenzo Gulisano, Marina Papatriantafilou, and Ashok Chaitanya Koppisetty. 2020. DRIVEN: A Framework for Efficient Data Retrieval and Clustering in Vehicular Networks. *Future Generation Computer Systems* 107 (2020), 1–17.

[18] Melanie Herschel, Ralf Diestelkämper, and Houssem Ben Lahmar. 2017. A Survey on Provenance: What for? What Form? What From? *VLDB Journal* 26, 6 (2017), 881–906. https://doi.org/10.1007/s00778-017-0486-1

[19] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep Learning. *Nature* 521, 7553 (May 2015), 436–444. https://doi.org/10.1038/nature14539

[20] Tom H. Luan, Longxiang Gao, Zhi Li, Yang Xiang, Guiyi Wei, and Limin Sun. 2015. Fog Computing: Focusing on Mobile Users at the Edge. *arXiv:1502.01815 [cs]* (Feb. 2015). arXiv:1502.01815 [cs] http://arxiv.org/abs/1502.01815

[21] Hannaneh Najdataei, Yiannis Nikolakopoulos, Marina Papatriantafilou, Philippas Tsigas, and Vincenzo Gulisano. 2019. STRETCH: Scalable and Elastic Deterministic Streaming Analysis with Virtual Shared-Nothing Parallelism. In *Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems - DEBS '19*. ACM Press, Darmstadt, Germany, 7–18. https://doi.org/10.1145/3328905.3329509

[22] Yiannis Nikolakopoulos, Marina Papatriantafilou, Peter Brauer, Martin Lundqvist, Vincenzo Gulisano, and Philippas Tsigas. 2016. Highly Concurrent Stream Synchronization in Many-Core Embedded Systems. In *Proceedings of the Third ACM International Workshop on Many-Core Embedded Systems - MES '16*. ACM Press, Seoul, Republic of Korea, 2–9. https://doi.org/10.1145/2934495.2934496

[23] Dimitris Palyvos-Giannas, Vincenzo Gulisano, and Marina Papatriantafilou. 2018. GeneaLog: Fine-grained Data Streaming Provenance at the Edge. In *Proceedings of the 19th International Middleware Conference (Middleware '18)*. ACM, New York, NY, USA, 227–238. https://doi.org/10.1145/3274808.3274826

[24] Dimitris Palyvos-Giannas, Vincenzo Gulisano, and Marina Papatriantafilou. 2019. GeneaLog: Fine-grained Data Streaming Provenance in Cyber-Physical Systems. *Parallel Comput.* 89 (Nov. 2019), 102552. https://doi.org/10.1016/j.parco.2019.102552

[25] Dimitris Palyvos-Giannas, Vincenzo Gulisano, and Marina Papatriantafilou. 2019. 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 (DEBS '19)*. ACM, Darmstadt, Germany, 19–30. https://doi.org/10.1145/3328905.3329505

[26] Dimitris Palyvos-Giannas, Bastian Havers, Marina Papatriantafilou, and Vincenzo Gulisano. 2020. Ananke: A Streaming Framework for Live Forward Provenance. *Proceedings of the VLDB Endowment* 14, 3 (2020), 391–403.

[27] Dimitris Palyvos-Giannas, Gabriele Mencagli, Marina Papatriantafilou, and Vincenzo Gulisano. 2021. Lachesis: A Middleware for Customizing OS Scheduling of Stream Processing Queries. In *Proceedings of the 22nd International Middleware Conference (Middleware '21)*. Association for Computing Machinery, New York, NY, USA, 365–378. https://doi.org/10.1145/3464298.3493407

[28] Thao N. Pham, Panos K. Chrysanthis, and Alexandros Labrinidis. 2016. Avoiding Class Warfare: Managing Continuous Queries with Differentiated Classes of Service. *The VLDB Journal* 25, 2 (April 2016), 197–221.

[29] Eugenia Politou, Efthimios Alepis, and Constantinos Patsakis. 2018. Forgetting Personal Data and Revoking Consent under the GDPR: Challenges and Proposed Solutions. *Journal of Cybersecurity* 4, 1 (Jan. 2018). https://doi.org/10.1093/cybsec/tyy001

[30] David Reinsel, John Gantz, and John Rydning. 2018. The Digitization of the World from Edge to Core. *Framingham: International Data Corporation* 16 (2018), 28.

[31] Ivo Santos, Marcel Tilly, Badrish Chandramouli, and Jonathan Goldstein. 2013. DiAl: Distributed Streaming Analytics Anywhere, Anytime. *Proceedings of the VLDB Endowment* 6, 12 (Aug. 2013), 1386–1389. https://doi.org/10.14778/2536274.2536322

[32] Mahadev Satyanarayanan, Pieter Simoens, Yu Xiao, Padmanabhan Pillai, Zhuo Chen, Kiryong Ha, Wenlu Hu, and Brandon Amos. 2015. Edge Analytics in the Internet of Things. *IEEE Pervasive Computing* 14, 2 (April 2015), 24–31. https://doi.org/10.1109/MPRV.2015.32

[33] Salman Taherizadeh, Andrew C. Jones, Ian Taylor, Zhiming Zhao, and Vlado Stankovski. 2018. Monitoring Self-Adaptive Applications within Edge Computing Frameworks: A State-of-the-Art Review. *Journal of Systems and Software* 136 (Feb. 2018), 19–38. https://doi.org/10.1016/j.jss.2017.10.033

[34] Valentin Tudor, Vincenzo Gulisano, Magnus Almgren, and Marina Papatriantafilou. 2018. BES: Differentially Private Event Aggregation for Large-Scale IoT-Based Systems. *Future Generation Computer Systems* (July 2018). https://doi.org/10.1016/j.future.2018.07.026

[35] Joris van Rooij, Vincenzo Gulisano, and Marina Papatriantafilou. 2018. 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 (DEBS '18)*. Association for Computing Machinery, New York, NY, USA, 171–182. https://doi.org/10.1145/3210284.3210298

[36] Blesson Varghese, Nan Wang, Sakil Barbhuiya, Peter Kilpatrick, and Dimitrios S. Nikolopoulos. 2016. Challenges and Opportunities in Edge Computing. In *2016 IEEE International Conference on Smart Cloud (SmartCloud)*. 20–26. https://doi.org/10.1109/SmartCloud.2016.18

[37] Ivan Walulya, Dimitris Palyvos-Giannas, Yiannis Nikolakopoulos, Vincenzo Gulisano, Marina Papatriantafilou, and Philippas Tsigas. 2018. Viper: A Module for Communication-Layer Determinism and Scaling in Low-Latency Stream Processing. *Future Generation Computer Systems* 88 (2018), 297–308. https://doi.org/10.1016/j.future.2018.05.067

[38] Nikos Zacheilas, Vana Kalogeraki, Yiannis Nikolakopoulos, Vincenzo Gulisano, Marina Papatriantafilou, and Philippas Tsigas. 2017. Maximizing Determinism in Stream Processing Under Latency Constraints. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems - DEBS '17*. ACM Press, Barcelona, Spain, 112–123. https://doi.org/10.1145/3093742.3093921