Open Access

Open Journal of Internet of Things (OJIOT)
Volume 6, Issue 1, 2020

http://www.ronpub.com/ojiot
ISSN 2364-7108

NebulaStream: Complex Analytics Beyond the Cloud

Steffen Zeuch[A,B], Eleni Tzirita Zacharatou[A], Shuhao Zhang[A], Xenofon Chatziliadis[A], Ankit Chaudhary[A], Bonaventura Del Monte[A], Dimitrios Giouroukis[A], Philipp M. Grulich[A], Ariane Ziehn[A], Volker Mark[A,B]

[A] Technische Universität Berlin, Straße des 17. Juni 135, 10623 Berlin, Germany
{firstname.lastname}@tu-berlin.de
[B] DFKI GmbH, Trippstadter Str. 122, 67663 Kaiserslautern, Germany {firstname.lastname}@dfki.de

Abstract

The arising Internet of Things (IoT) will require significant changes to current stream processing engines (SPEs) to enable large-scale IoT applications. In this paper, we present challenges and opportunities for an IoT data management system to enable complex analytics beyond the cloud. As one of the most important upcoming IoT applications, we focus on the vision of a smart city. The goal of this paper is to bridge the gap between the requirements of upcoming IoT applications and the supported features of an IoT data management system. To this end, we outline how state-of-the-art SPEs have to change to exploit the new capabilities of the IoT and showcase how we tackle IoT challenges in our own system, NebulaStream. This paper lays the foundation for a new type of systems that leverages the IoT to enable large-scale applications over millions of IoT devices in highly dynamic and geo-distributed environments.

Type of Paper and Keywords

Visionary paper: *IoT, application, systems, data management, stream processing, smart city*

1 Introduction

The volume, velocity, and variety of data that need to be processed in real-time have reached unseen magnitudes over the last decades. This trend fueled the emergence of the first stream processing engines (SPEs), such as Aurora, STREAM, TelegraphCQ, and NiagaraCQ, which process queries over incoming data streams continuously [16]. However, the ever-increasing input rates in combination with the demands on low-latency and high-throughput computing led to a new class of SPEs, called *Large-Scale Data Stream Processing Systems* [16]. Systems such as Flink, Spark Streaming, or Storm are representatives of this class. These systems scale-out the processing of high-velocity data streams in a cluster of commodity hardware. Although these systems enable large scale execution on hundreds or thousands of servers, they are most effective when data are generated and used within the cloud. However, transferring data from external sources into the cloud induces a major bottleneck that limits the scalability. Furthermore, sending computation results back to devices outside the cloud further exacerbates this bottleneck.

The arising Internet of Things (IoT) represents a

This paper is accepted at the *International Workshop on Very Large Internet of Things (VLIoT 2020)* in conjunction with the VLDB 2020 conference in Tokyo, Japan. The proceedings of VLIoT@VLDB 2020 are published in the Open Journal of Internet of Things (OJIOT) as special issue.

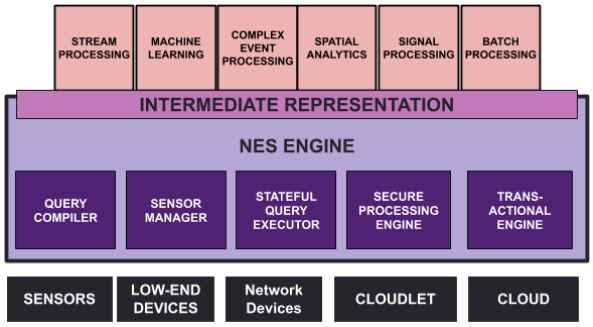


Figure 1: Complex Analytics in NES.

new environment that challenges the above scalability bottleneck as data are mainly generated outside the cloud and results must be sent back to cloud-external devices. In particular, the IoT adds further pressure on current SPE architectures as estimations predict that by 2025 the global amount of data will reach 175ZB and 30% of these data will be gathered in real-time [59] by as many as 20 billion connected devices [36]. As a result, the majority of these data will be generated and used by devices in highly-distributed, potentially geo-distributed, environments. However, state-of-the-art cloud-based SPEs are not built for such an environment and require data to be transferred into the cloud to extract value from them. In contrast, an IoT data management system will enable data processing along the path from data sources to the cloud, using the increasing capabilities of cloud-external devices.

Over the last decades, many research communities cover specific parts of the overall IoT vision. First, Mobile Cloud Computing (MCC) outsources data storage and processing from devices to the cloud [7]. Second, Mobile-Edge Computing (MEC) extends IoT services through hub devices at the edge of the topology, which act as local control centers [62]. Third, Fog-aware IoT data processing uses the *Fog*, a collective term for resources outside the cloud, as an extension of cloud [3, 63, 75, 10]. Finally, Sensor networks (SNs), in particular Wireless Sensor Networks (WSN) target distributed processing in a wireless network of sensors as a particular sub-area of the IoT [48]. With NebulaStream (NES), we are currently building a data management system for upcoming IoT environments that combines approaches from these research communities as well as the broad literature of cloud-based SPEs and introduces novel solutions [78]. In our vision, NES pioneers a new class of systems, that copes with heterogeneity and distribution of compute and data, supports diverse data and programming models going beyond relational algebra, deals with potentially unreliable communication, and enables constant evolution under continuous operation.

In Figure 1, we outline the component stack of NES. NES will support various algorithms from different application domains with different requirements and expressiveness. These algorithms are implemented on top of a common intermediate representation, which expresses the common set of functionality offered by NES. The NES Engine takes the queries expressed in the intermediate format as input and orchestrates their processing. To this end, the NES Engine consists of 1) a query compiler to generate executable code, 2) a sensor manager to interact with the attached sensors, 3) a distributed dataflow engine that efficiently executes long-running stateful queries, 4) a secure processing engine that ensures privacy-aware and secure processing, and 5) a transactional engine that orchestrates consistent transactional processing. The NES Engine runs on a variety of heterogeneous and diverse devices, e.g., sensors, low-end devices (e.g., Raspberry Pi), network devices (e.g., smart routers), cloudlets, or cloud nodes. With this component stack, the main goal of NebulaStream is to enable the emerging IoT applications of the future.

In this paper, we bring together the requirements of upcoming IoT applications and the supported features of an IoT data management system. While developing NES, we constantly review state-of-the-art approaches to identify which concepts and algorithms can be adopted and for which problems we require novel solutions. As our main IoT application, we target a smart city, where millions of sensors are distributed across the city to gather information about traffic, air and water quality, crowdedness, and many more. In the following paper, we first describe core features that are necessary to enable the next generation of IoT applications for smart cities but are not yet supported by state-of-the-art systems, i.e., adaptive sensor handling (Sec. 2.1), massive scalability (Sec. 2.2), heterogeneity of workloads and devices (Sec. 2.3), and delivery guarantees (Sec. 2.4). After that, we investigate two new features that will enable new classes of applications within a smart city but are currently neither part of common SPEs nor IoT data management systems, i.e., secure-privacy-aware (Sec. 2.5) and transaction processing (Sec. 2.6). Finally, we investigate domain-specific features that can support various application scenarios within a smart city, i.e., support for signal processing (Sec. 3.1), spatial analytics (Sec. 3.2), complex event processing (Sec. 3.3), and machine learning (Sec. 3.4). For all presented features, we outline the challenges and requirements, discuss the state-of-the art, show their limitations, and highlight how to efficiently use them in IoT infrastructures. Overall, this paper lays the foundation for a new type of SPEs that leverages the IoT to enable new possibilities for

application scenarios such as smart cities.

2 CORE FEATURES

In this section, we describe core features that are required to enable the next generation of IoT applications but are not yet supported by state-of-the-art systems.

2.1 Adaptive Handling of Sensor Data Streams

Sensor data in IoT environments have a fluctuating nature, which poses a challenge for resource-constrained devices that cannot easily cope with the velocity and volume of incoming data. Overwhelmed devices incur back-pressure, which further results in data losses and high query latency. To address this challenge and avoid performance deterioration, an IoT system needs to adapt the number of sensor reads to the observed phenomenon and maximize sharing of sensor results, thereby avoiding resource overuse.

2.1.1 State-of-the-Art Systems

Current state-of-the-art systems focus on handling processing at the edge and under a *homogeneous hardware* assumption [56], i.e., homogeneous networking equipment or execution nodes. Other state-of-the-art SPEs deal with stream fluctuation by employing *input admission* [9], where systems keep in check incoming data streams, but forego any optimizations at the source level. Finally, well-known sensor-based systems focus on disseminating a single query over a sensor network [48, 74].

2.1.2 Limitations of State-of-the-Art

In the IoT, streams are ephemeral and start from the level of sensors. Sensors are distributed, transient, prone to failure, resource-constrained, and offer a diverse feature set that is heterogeneous among nodes. Moreover, applications require sensor data under multiple assumptions and constraints, e.g., outlier detection requires real-time data but monitoring trends over large time-series requires low sampling and no filtering.

Current state-of-the-art systems either deal with specific parts of an IoT environment or assume the existence of homogeneous resources. At the same time, systems that only focus on sensors do not take the increased capabilities of the nodes that host the sensors (sensor nodes) into account and cannot be easily integrated into existing SPEs. In general, existing systems do not treat sensor nodes as a first-class component of an SPE and, thus, do not allow for applications to benefit from any optimizations on that particular level.

Supporting adaptive operations at the sensor node level allows one to scale the number of sensors while reducing the volume of network traffic without harming the precision of the results. The system will be able to distinguish *interesting* events while allowing only transmission of *important* data.

2.1.3 Enabling Emerging IoT Applications

IoT applications, such as real time monitoring of drone fleets or outlier detection of sensor data on a city-wide infrastructure, produce workloads of fluctuating nature. This causes local bottlenecks that need to be dealt as early as possible, before they propagate downstream. With NES, we envision a true end-to-end system, where fluctuation mitigation starts at the sensor nodes. By utilizing *adaptive sampling* and *adaptive filtering* of sensor data at the source, we keep in check the dynamicity of various workloads without altering the context of the data. Through these operators, we are able to *a) conserve energy* and *b) reduce unnecessary network communication* while *c) retaining a high-quality representation* of the original data stream.

2.2 Massive Scalability

An IoT data management system continuously receives various machine-generated or user-submitted long-running and ad-hoc queries. To be effective, it needs to provide support for millions of concurrent queries [70]. In addition, an IoT system needs to handle millions of highly heterogeneous and distributed data streams that can be achieved by routing the streams as efficiently as possible through the network of processing nodes.

2.2.1 State-of-the-Art Systems

Several approaches for processing IoT data exist today. A cloud-based system such as Mobile Cloud Computing (MCC) relies on gathering and sending the data from a pool of sensors to cloud for storage and processing [7]. Cloud infrastructures make use of common SPEs, such as Apache Flink [17], Spark Streaming [77], or Storm [68] for processing the incoming data streams.

Edge-aware systems such as Mobile-Edge Computing (MEC) overcome the limitations of cloud-centric approaches by utilizing *hub devices* to extend their IoT services [62, 6]. Hub devices reside at the edge of the fog topology for gathering data from attached sensors and performing simple processing steps.

Fog-aware systems, e.g., Frontier [56], Disco [11], or the extension of Cisco's Connected Streaming Analytics

(CSA) [62] utilize the processing capability of the fog by distributing queries over the topology.

2.2.2 Limitations of State-of-the-Art

Existing approaches for processing IoT data have several limitations. The cloud-based approach is limited by the amount of data it can receive from IoT devices. A higher number of IoT devices results in a higher volume of data, which can potentially create a network bottleneck and thus create delays in the processing. Similarly, an edge-aware approach that uses hub devices has the limitation in terms of performing holistic data processing. Also, this approach does not leverage the cloud resources for performing additional computations. This affects the type of queries that are supported by the system. Additionally, the hub device controls the number of IoT devices that it can support and thus restricts the scalability. Fog-aware approaches mentioned above have limitations in handling evolving infrastructure. This restricts the processing of queries in a large dynamic environment where the IoT devices are constantly joining and leaving the environment.

NES considers a unified fog and cloud environment for processing IoT data. This enables NES to perform early computations in fog and any remaining or holistic computations in the cloud. Additionally, the support for sustaining network or node failures and device mobility allows NES to handle evolving topology.

2.2.3 Enabling Emerging IoT Applications

NES will efficiently process large volumes of data coming from millions of devices in a dynamic environment. This will enable NES to support a new generation of applications spanning over millions of geo-distributed devices across a smart city, such as connected cars or smart public transport systems. Furthermore, efficient query execution allows for handling massive amount of queries generated by external systems or submitted by users.

2.3 Support for Heterogeneous Devices

IoT environments consist of a wide range of diverse compute devices (from small sensors to high-performance data centers). Sensors that generate the source data stream are often battery-powered and have typically limited computing resources. In contrast, the cloud has nearly unlimited compute and storage resources, while cloud nodes are equipped with high-performance multi-core CPUs and accelerators like GPUs and TPUs. The fog represents a suitable middle-ground between the sensors and the cloud, since it contains more capable nodes than sensors. These nodes consist of embedded System-on-a-Chip Devices (SoCs), low-energy servers, or contain specialized accelerators like embedded GPUs (e.g., Nvidia Jetson) or TPUs (e.g., Coral Edge TPU). As a result, an IoT data management system needs to support an unseen variety of heterogeneous devices, which it must efficiently exploit. This heterogeneity introduces several challenges, as devices use different data formats (little- vs. big-endian), different programming models (CPUs vs. GPUs), and different instruction sets (ARM vs. x64). Depending on the workload, the system has to pick the right device and accelerator to meet the desired performance and energy requirements.

2.3.1 State-of-the-Art Systems

Current data processing systems usually choose between two extreme design choices. General-purpose SPEs are usually *hardware-oblivious*, e.g., Flink [17], Spark Streaming [77], or Storm [68]. They use virtual machines, e.g., the Java Virtual Machine, to abstract from the underlying hardware. This allows the support of a wide range of computing devices. However, it limits the efficiency as specific hardware characteristics are not utilized. In contrast, other SPEs are *hardware-specific*, e.g., TinyDb [48], Streambox-TZ [57], Grizzly [34], or BriskStream [81], and optimize for very specific hardware characteristics. However, they cannot generalize across a diverse set of devices with heterogeneous hardware capabilities. As a result, no state-of-the-art system provides a system architecture that can support a wide range of diverse devices and, at the same time, exploits specific hardware characteristics, if available.

2.3.2 Limitations of State-of-the-Art

The IoT has fundamental challenges that make current data processing solutions insufficient. First, devices are very diverse, which requires system support for a wide range of different hardware characteristics. Second, the devices in the IoT have limited hardware resources and energy budgets, which make abstractions, e.g., a virtual machine, not feasible. To this end, NES is *heterogeneous hardware-conscious* and optimizes both requirements in a holistic and general way.

2.3.3 Enabling Emerging IoT Applications

The support of diverse devices is a core requirement of every SPE in an IoT environment. The efficient exploitation of the individual hardware resources of such devices is crucial for enabling a range of new innovative applications.

One example of such an application is a smart public transport monitoring system [31]. Such systems gather data from different sources and process it across diverse compute nodes. For instance, vehicles collect the source data and use small battery-powered SoCs to perform preprocessing. Using a wireless connection, data are transferred to intermediate base stations, which may perform further calculations. As soon as the data reach the cloud, the SPE computes a final result. The above layers consist of very diverse hardware and have different requirements. NES takes this heterogeneity into account to generate highly efficient code to run in each layer.

2.4 Delivery Guarantees

IoT applications require diverse guarantees for record delivery and state update semantics. These semantics affect the consistency of query results and are categorized as (a) *at-most-once* i.e. record losses are allowed, (b) *at-least-once* i.e. record duplicates are allowed, and (c) *exactly-once* i.e. every record must be processed only once.

2.4.1 State-of-the-Art Systems

State-of-the-art, cloud-based SPEs support exactly-once, at-most-once, and at-least-once state updates [15, 22, 5]. They assume reliable, TCP-based network connections that do not lose data. Furthermore, they assume *upstream backup* [37] and sophisticated recovery mechanisms [15, 5] via persisted state checkpoints to prevent record and state loss.

2.4.2 Limitations of State-of-the-Art

In an IoT environment, the assumption of reliable network connections no longer holds. Furthermore, upstream backup and state checkpoints are not feasible as the underlying infrastructure is volatile and unreliable. As a result, IoT applications will perform inefficiently and be error-prone if the above issues are not solved. Furthermore, deterministic computation (see Section 2.6) is not feasible, if there is no reliable partial order-preserving delivery guarantee. Overall, enabling reliable stream processing on a highly unreliable infrastructure of low-end devices is an open research question. In particular, the level of consistency achievable and the trade-off between consistency and performance must be carefully investigated.

2.4.3 Enabling Emerging IoT Applications

Supporting delivery guarantees enables applications to enforce or relax constraints on record processing. Delivery guarantees are not a new concept in stream processing. However, the volatility of the IoT makes them particularly hard to provide. We envision to address this challenge with NES, by allowing applications to trade performance for consistency or vice versa, depending on the use case. In particular, applications that do not need tight constraints on results may use *at-most-once* semantics and allow record loss to speed up analytics, e.g., on temperature reads. However, applications that deal with sensitive data, e.g., car accident detection, may employ *at-least-once* or *exactly-once* semantics at the expense of latency.

2.5 Secure & Private Stream Processing

There is a fast increasing volume of data generated in the IoT environment. To achieve timely analysis, rather than processing in the cloud, data must be processed near the source (e.g., the cloud edge) as much as possible to reduce transmission overhead. However, the above edge processing scheme exposes sensitive data to significant security threats as edge devices are vulnerable to being attacked, causing information leakage. *Secure & private stream processing* aims to provide a built-in security protection mechanism for SPEs, which has to be both scalable and energy efficient. Existing mechanisms such as Trusted Execution Environment (TEE) and homomorphic encryption (HE) may require a revisit in order to be applied in an IoT environment.

2.5.1 State-of-the-Art Systems

The Trusted Execution Environment (TEE) technique is one promising approach to secure data processing. It isolates a special encrypted area of memory called an enclave. Subsequently, it guarantees that code and data loaded in the enclave are protected with respect to confidentiality and integrity. Park et al. [57] reported how to extend existing SPEs [52] to utilize ARM TrustZone, which is one of the implementations of TEE. However, it still remains unclear how to efficiently support stateful applications, in particular when the enclave can not hold large application states. For example, it can only hold up to tens of megabytes (MB) for ARM TrustZone and up to 128 MB for Intel SGX enclave [13].

Another promising approach is to utilize a homomorphic encryption (HE) mechanism [14], which allows directly processing on encrypted data (i.e., cyphertext) without decryption. However, how to efficiently utilize homomorphic encryption mechanisms in SPEs still remains an open question. On the one hand, partial HE mechanisms, which allow for a restricted set of operations (e.g., ordering) on cyphertext, may still

lead to information leakage. For example, CryptDB [58] utilizing order-preserving encryption is reported to be easily crackable [30]. On the other hand, all known fully HE schemes, which support any operations on encrypted datasets and provide more reliable security guarantees, still have a long way to go before they can be used in practice due to the significant computational complexity [54]. The strict low-latency processing requirement of SPEs and the low processing capability of IoT devices prohibit the use of any computationally heavy encryption scheme.

There exists an extensive body of literature focusing on privacy preservation in relational databases, e.g., k-anonymity [66] and differential privacy [25]. However, both approaches focus on statistical queries on static relational data. They are not directly applicable to stream processing, which has to achieve low processing latency while dealing with continuous input streams. Specific solutions have also been designed for privacy-aware stream processing, such as privacy-aware complex event processing [35]. However, how best to design a general-purpose privacy-aware SPE still remains an open question. Unstable network connections and potentially noisy data sources in the IoT environment bring even more challenges to the privacy-aware system design. For example, it becomes hard to tell whether the noise [25] or suppression [35] is introduced by the privacy protection mechanism or by the dynamic IoT environment.

2.5.2 Limitations of State-of-the-Art

The shift from powerful servers in a cloud environment to low power IoT devices requires us to revisit existing security mechanisms. On the one hand, providing security guarantees often requires one to trade-off performance as it brings significant computation complexity. On the other hand, IoT devices are often equipped with relatively low computation capacity with unstable connections.

The federated environment of IoT also prohibits us from relying on centrally governed mechanisms to provide security guarantees. For example, an application may require data streams generated across different regions (e.g., data from the capital of different countries). However, sensitivity information may not be allowed to leave a specific region. However, existing mechanisms are mostly non-scalable when we have to provide security guarantees to millions of devices across different layers (i.e., edge, fog, and cloud center) in a large geographic scale.

2.5.3 Enabling Emerging IoT Applications

Secure & Private stream processing enables NES to process confidential data by providing security and integrity guarantees. This allows NES to be adopted into many sensitive aspects of a smart city such as smart health care, and smart finance, where data may be generated over millions of sensors, and privacy and integrity need to be protected. For example, heart rate variability analysis [61] requires analyzing electrocardiograms (ECG) and photoplethysmograms (PPG). Both signals must be captured and processed reliably in real time. It is challenging in supporting this use case. On the one hand, any unauthorized read or even modification to the input signals or application states (e.g., range of typical values of signals) can be dangerous and should be prevented. On the other hand, large processing latency can cause delays in identifying emerging situations. To minimize network transmission overhead, we need to process the data close to the source (e.g., sensors on the patients), which is however vulnerable to be attacked. NES aims to bring security guarantee to stream processing without introducing significant overhead. In particular, we will take a holistic approach by applying suitable solutions on different components of stream processing across different layers.

2.6 Transactional Stream Processing

SPEs with transactional state management relieve the burden of managing state consistency from the users. However, scaling stream processing while providing transactional state management is challenging, in particular for emerging dynamically distributed environment such as the IoT. On the one hand, to achieve both low latency and high throughput, SPEs can process multiple input events (including streaming data and ad-hoc user queries) [42] at the same time to aggressively exploit parallelism. On the other hand, processing different events concurrently may lead to conflict accesses (reads and writes) to the same application state (i.e., *concurrent state access*), hence leading to higher chances of violating the transactional state consistency [51, 83]. Furthermore, more than simply guaranteeing the ACID properties, SPEs need to enforce the state access order according to the input event sequence. This is very different from the conventional concurrency control protocols, which serialize transactions in an order that is conflict-equivalent to any certain serial schedule.

2.6.1 State-of-the-Art Systems

In the area of general stream processing engines, recently proposed SPEs achieve excellent performance when processing large volumes of data under tight latency constraints [79]. In particular, SPEs such as Flink [17], Storm [68], and Heron [46], achieve high performance via disjoint partitioning of application states [15] – often through hash partitioning [43]. This ensures that each execution thread (i.e., executor) maintains a disjoint subset of states and thereby bypasses the issue of *concurrent state access*. However, this type of design can lead to tedious implementation and ineffective performance in many cases [83].

Transactional stream processing engines have recently received attention from both academia [2, 83] and industry [27]. In emerging use cases, large mutable application states can be concurrently accessed by multiple operators (and their executors) [83, 51, 2] during stream processing and transactional state consistency has to be preserved. Many such applications have been proposed covering various domains (e.g., Health-care [71], IoT [12], and E-commerce [51]). Recent transactional SPEs typically model state accesses as transactions [12]. Subsequently, state consistency is maintained by the system by adopting transactional semantics (such as ACID properties). However, they typically rely on locks, where concurrent access to each state is permitted only if the lock is granted to the thread. As a result, it scales poorly while underutilizing hardware resources [12, 51]. TStream is a recently proposed transactional SPE [83], and it significantly improves the execution efficiency by completely avoiding locks. However, it is designed for a single node assuming a shared-memory architecture. It might require a system redesign to fully take advantage of TStream's approach in an IoT environment.

2.6.2 Limitations of State-of-the-Art

A straightforward mechanism to support transactional stream processing is to adopt an off-the-shelf transactional database management system (DBMS) for state maintenance during stream processing. Unfortunately, this mechanism not only degrades the system performance but may also violate state consistency [51]. On the one hand, using a third-party DBMS for frequent data access can cause high inter-process communication overhead between two different systems (i.e., DBMS and SPE). On the other hand, conventional DBMSs' concurrency control protocol only guarantees that the execution order of concurrent transactions is conflict-equivalent to any certain serial schedule, which may not obey the event order implied by the attached timestamps [51, 83].

Furthermore, a central state storage is often not available in an IoT environment. This brings even more challenges to provide state consistency as transactional states have to be stored in a distributed manner. Although distributed databases/key-value stores have been extensively studied and adopted in some SPEs, they are not designed to support concurrent ordered state accesses [83, 2]. Guaranteeing a deterministic execution sequence and state consistency while supporting concurrent state access is also in contradiction to the highly dynamic IoT environment where input record can arrive out of order, have an inaccurate timestamp, and miss information. Common problems in the IoT environment, e.g., transient node errors, unreliable connections, low-quality hardware, low storage capacities, and no upstream backups, further make existing scale-up [83] and scale-out [2] solutions hardly applicable.

2.6.3 Enabling Emerging IoT Applications

Transactional stream processing opens the gate of linking two popular research fields, OLTP and stream processing. It eases the development of many emerging complex stateful stream applications [83], where the processing of a single event may need to access multiple overlapping states while preserving state consistency. Let us take self-driving vehicle monitoring [50] as an example, which is one of the popular applications of a smart city. Millions of cars are continuously generating status data via their sensors, and SPEs can offer many services, such as a warning and a list of nearby gas stations whenever the fuel tank level is below a certain threshold. The SPE then needs to maintain gas station information, road condition information, while processing the flood of sensor data streams from vehicles. To make the right decisions timely, *consistent and up-to-date* states of both gas stations and roads is crucial in this example. This is challenging because the processing of input signals from different vehicles may access common application states, e.g., status of a common gas station or road. To relieve users from managing shared state consistency by themselves, NES will provide an efficient transactional state management component to perform data integration, analytics, cleaning, and transformation on fresh data in near real time.

3 Domain-Specific Features

In this section, we describe domain specific features that are required to enable a richer set of applications over an IoT data management platform such as NES.

3.1 Signal Processing

Networked sensors are at the core of the IoT. To analyze sensor data and extract insights, IoT applications employ a variety of signal processing techniques, e.g., interpolation to handle missing sensor values, digital filters to recover noisy signals, and Fast Fourier Transform (FFT) to do spectral analysis [55]. In addition, applications need relational operators, e.g., filters to select data subsets, joins to combine signals with external data sources, or group-by aggregates to group signals according to their source.

Domain experts typically use numerical frameworks such as Matlab or R to perform signal processing. However, these frameworks lack support for relational operators, are not suited for online analyses, and cannot scale to large datasets. To address these limitations, big data frameworks, such as Spark, integrate with R, allowing domain experts to re-use their R scripts, which are now executed on Spark's processing engine. Nevertheless, there is an impedance mismatch between general-purpose data processing systems and numerical frameworks, which introduces high communication overhead, making it hard to attain real-time performance.

3.1.1 State-of-the-Art Systems

Conventional SPEs do not have built-in signal processing support. A naive solution to enabling digital signal processing (DSP) is to implement signal processing operations as user-defined functions. This is, however, counter-intuitive for domain experts, as it requires a deeper understanding of the system's data model.

There is a limited amount of work in unifying relational and signal processing operations into a single system (e.g., WaveScope [32] and TrillDSP [55]). WaveScope provides little support for distributed query execution. It executes distributed applications over a cluster of a few processing nodes or between processing nodes and sensors. Only lightweight operations are offloaded to sensors, while a centralized component collects data for further processing. TrillDSP extends Microsoft's Trill streaming analytics engine [18], which does not support distributed execution.

3.1.2 Limitations of State-of-the-Art

There are only a few stream processing engines that provide adequate support for DSP operators [32, 55]. However, these engines have no or little support for distributed query execution. Therefore, they cannot be deployed in a highly distributed IoT environment. Enabling signal processing over streams in such a heterogeneous environment is challenging for several reasons. First, some DSP operations are compute or memory intensive, and thus cannot be executed on resource-constrained, low-end fog devices. At the same time, there might be devices close to the input sensor sources that are equipped with specialized signal processing hardware and are thus particularly tailored for efficiently executing DSP operators. Furthermore, DSP operations are typically not commutative. Therefore, event-ordering has to be maintained, which is hard in a highly volatile and distributed fog infrastructure. NES aims to provide efficient native support for signal processing by tightly integrating DSP operators in the execution engine, performing semantic-aware data routing, and implementing smart operator placement strategies to optimize resource utilization.

3.1.3 Enabling Emerging IoT Applications

Sensors capture and transmit signals in streams. To analyze these data, IoT workflows in various domains combine relational and signal logic. ShotSpotter [64], for example, is a gunshot detection and localization application that captures impulsive audio signals likely to correspond to gunshots using acoustic sensors placed in a city. The signals are grouped by regions and are filtered to remove background noise, prior to being further processed by sophisticated detection techniques. NES aims to support the execution of DSP operators close to the sensors, and thereby enable emerging IoT applications to efficiently process high-rate signals originating from millions of sensors.

3.2 Efficient Spatial Analytics

Data produced by IoT devices are inherently geospatial in nature. Some devices have fixed static locations (e.g., smart house appliances), while others (e.g., smartphones, smartwatches, wearables) have a continuously changing location, as they are attached to a moving entity (e.g., a human or a car). The volume and rate of geospatial data collected in the IoT are ever-increasing. Cellular networks produce millions of records per second [38]. Service weather stations have a broad range of sensors that measure atmospheric conditions with increasing granularity, generating millions of records with each scan. At the same time, a variety of applications in domains such as transportation, environmental sciences, health, and public safety rely on efficient, real-time spatial data processing. Connected vehicle applications leverage spatial data to seamlessly optimize mobility in smart cities. Monitoring systems require fresh spatial data to prevent dangerous situations and trigger alerts. Given the central role that spatial information plays in the IoT and the interactivity expected from applications, there is the need for an IoT data management platform

that supports low-latency spatial analytics.

3.2.1 State-of-the-Art Systems

In the area of cluster-based systems, most existing distributed spatial systems are designed for static (batch) processing, and thus incur high latency for streaming queries over streaming data. They are based on Hadoop [4, 26] or Spark [67, 72, 76].

Existing general-purpose SPEs [17, 68] have no direct support for spatial data. MobyDick [29] is a library built on top of Apache Flink [17] that extends Flink with a set of spatio-temporal data types and operators. Tornado [49] extends Apache Storm [68] to support continuous spatial-keyword queries. These extensions are initial research prototypes supporting only a limited set of spatial query types.

3.2.2 Limitations of State-of-the-Art

Existing systems face many challenges in supporting spatial analytics in the IoT. First, they need to collect all data in a centralized cluster or cloud environment prior to applying processing. Given the large amounts of data originating from millions of geo-distributed sensors, this centralized processing paradigm results in high query latency. Second, data are multidimensional, heavily skewed, and come at high velocity. As a result, load balancing and indexing techniques employed by existing spatial data frameworks either incur high ingestion rates due to frequent re-balancing operations or penalize query performance due to partitioning schemes that do not preserve spatial locality [39]. To achieve low-latency and scalability, NES will employ novel solutions that process spatial data in-network, in a locality-aware manner. Finally, spatial analytics involve costly geometric computations (such as Point-in-Polygon tests) to evaluate relations between objects in space (e.g., intersection, containment) and are thus computationally expensive [44, 65, 69], more expensive than typical relational queries. Low-end IoT devices have limited computational resources, and are thus unsuitable for executing such costly operations. To make efficient use of the available resources, NES will employ smart placement strategies that distribute spatial queries across fog and cloud nodes while exploiting specialized hardware (e.g., GPUs) and respecting device limitations.

3.2.3 Enabling Emerging IoT Applications

Most phenomena in the IoT are location-dependent. Consequently, the relevance, value, and utility of IoT data typically depend on the geographic context of the devices that produce data and of the users that consume them. NES aims to provide efficient, real-time spatial data analytics, and thereby facilitate emerging IoT applications in domains such as transportation (e.g., driverless vehicles, connected cars), public safety (e.g., a network of connected cameras or acoustic sensors) and health (e.g., wearable health trackers)

3.3 Complex Event Processing

The IoT is one of the main domains that successfully leverage the monitoring features of Complex Event Processing (CEP) [21, 82]. CEP is a stateful stream processing method that detects user-defined rule patterns in large data streams. Herewith, it enables autonomous real-time decision making for data-intensive IoT applications where manual monitoring is infeasible and prompt reactions are required, e.g., intelligent transportation systems, smart street lamps, vehicle pollution control, or supply chain management [3]. Thus, efficient CEP is a necessary feature for future IoT applications with millions of connected devices.

3.3.1 State-of-the-Art Systems

Several SPEs offer CEP features, e.g., Esper[1], Cayuga [23], STREAM [8], and Aurora [1]. Due to the ever-increasing volume and rate of data, the concurrent detection of thousands of patterns requires massive resource capacities. To efficiently utilize the existing resources of these mostly single-machine approaches, optimization techniques can be applied, e.g., rewriting and prefix sharing [45, 60]. However, the majority of these systems use pattern specification languages, which make automated optimization of their pattern detection mechanism challenging [60].

Examples of Large-Scale Data Stream Processing Systems [16] that offer CEP are Flink [17], Spark [77], and Storm [68]. These systems are often cloud-based and, thus, provide almost unlimited resources. Parallel stream processing and distributed pattern detection monitoring are techniques to distribute the centrally collected data and efficiently utilize these unlimited cloud resources [28]. However, these techniques do not solve the central data collection bottleneck.

Overall, CEP consists of two components: a pattern specification language (to define complex event patterns) and a pattern detection mechanism (to detect patterns in data streams). For neither of the two components, a general solution exists, i.e., several CEP systems provide their own specification languages (e.g., EPL[1], SASE+ [80], and CCL [82]) optimized for their pattern detection mechanism. Detection mechanisms typically focus on either several variations of state machines or

[1] http://www.espertech.com/

trees, but also event processing networks and column-based approaches exist [45]. The missing generality of CEP systems leads to various approaches with individual optimizations that are hard to adapt to another CEP system or computing paradigm [45, 84]. Besides, these variations strengthen the challenge of coping with the rapid evolution of future IoT applications.

3.3.2 Limitations of State-of-the-Art

The central data collection bottleneck of current SPEs prevents their CEP engine from exploiting the spatial closeness of event sources as well as data reduction and distributed pattern detection close to the network's edge [19, 20]. To this end, current SPEs are not ready yet to provide CEP solutions that fulfill the low-latency and real-time requirements of IoT applications for millions of distributed IoT devices.

Fog environments enable data processing at the edge of the network using geographically distributed low-end devices. As a result, computational and time-intensive cloud-based pattern detection mechanisms cannot be applied out-of-the-box and require adjustments to fit the hardware capabilities and be network-aware.

Future data management platforms need CEP to enable autonomous monitoring applications for the IoT. Within NES, we want to provide a distributed in-network CEP for future IoT applications by exploiting the fog paradigm given limitations of the low-end devices and dynamic network topology.

3.3.3 Enabling Emerging IoT Applications

CEP is a powerful stream processing method, commonly used by various monitoring applications. By adapting current CEP systems with enhanced distribution strategies beyond cloud solutions, NES will provide CEP also for future IoT monitoring applications. These monitoring applications profit further from the previously introduced features, i.e., spatial data analysis for location-aware patterns, adaptive filtering for early data reduction as well as secure and transactional processing techniques. In particular, the union of the cloud and fog paradigms in a single environment will allow applications to control the reporting of partial and full matches flexibly to the interested sinks. Some applications may run entirely autonomously in the fog layer, e.g., smart street lamp monitoring [40]. Other applications might expose their matches to the cloud layer, but their sensitive data, which was analyzed to detect the match, will not leave a private fog and is therefore secured. For example, smart hospitals [73] reporting COVID-19 cases to a smart city application for further analysis. Overall, distributed CEP for the IoT enables a wide range of autonomous monitoring applications that are currently prevented by cloud bottlenecks and data security concerns.

3.4 Machine Learning

Complex machine learning (ML) tasks, e.g., classification, clustering, and prediction are key applications to extract knowledge from massive amount of IoT data. Therefore, it is necessary to provide native ML support in an IoT data management platform to fully profit from the insights extracted from the data.

3.4.1 State-of-the-Art Systems

As presented by Derakshan et al. [24], a machine learning model training pipeline consists of several stages. A pipeline is iterative as every stage is repeated to improve the performance of the model continuously. The key components for each stage of such a pipeline are the following: (1) Definition of the input sources, (2) Data cleansing and transformation, (3) Training, (4) Evaluation, (5) Materialization, (6) Deployment.

State-of-the-art SPEs, e.g., Flink do not support all stages natively. In most application scenarios only Stage (1) and (2) are executed in the SPE. For training and evaluation (Stage 3 and 4), the data is then loaded to a batch processing system like Spark or Tensorflow. After the model is trained, it is transmitted in Stage 5 to a ML model management system like MLFlow [53] or ModelDB [47] At this point, the model is persisted at a central location, and a versioning number along with additional metadata, is attached to it. The deployment of the model is performed in Stage 6, where all destinations that require the newest version of the trained model can access it from the centrally located model management system in the cloud.

In situations where training or inference is required within the SPE, users have to implement customized solutions using User Defined Functions (UDFs). In these cases, the processes for inference are running as micro-services in a central location and are called via REST or Remote Procedure Calls (RPCs).

3.4.2 Limitations of State-of-the-Art

Pushing training and inference operators down to edge devices is challenging from a system perspective, mainly due to the complexity of ML pipelines. As described above, there is no system supporting all the needed functionality for all stages of the pipeline.

In the IoT domain, a data management system for ML should also cope with the following three characteristics of IoT topologies and their corresponding challenges. First, *heterogeneity*: The training of a

model is a computationally expensive operation that requires many resources and is usually executed on GPUs. Consequently, model training tasks should be preferably executed in the cloud. Devices in the IoT vary in resources and run on different operating systems and hardware. Applying ML in such a heterogeneous IoT environment is currently not supported by common ML Systems like Tensorflow or Spark. In particular, specialized approaches for ML inference in IoT environments [41, 33] are not supported by a general purpose systems for ML.

Second, *limited resources*: Difficulties arise from the large sizes of the trained models. The states of operators that incorporate pre-trained ML models usually reach sizes of several gigabytes. Handling states of this magnitude is challenging for memory-constrained IoT devices. Additionally, distributed training of an ML model requires many synchronization messages across all devices. Given that typical IoT topologies consist of several thousands of edge devices, this becomes problematic in a setting where network bandwidth is limited. Third, *unreliability*: A foundational problem in the context of streaming in the IoT is also the way data is transmitted to the different devices. Applying existing cloud-based solutions in a fog infrastructure risk to stall the whole system, as these solutions are prone to backpressure or high latency. In application scenarios with long-running ML pipelines, it is thus essential to handle these issues, as they could prohibit the completion of the pipeline.

With NES, we propose a system that will support processing tasks on CPUs and GPUs in the cloud, fog, and edge. As a result, NES eliminates the need to perform all pipeline stages in the cloud and thus enables more efficient execution of ML pipelines.

3.4.3 Enabling Emerging IoT Applications

Nowadays, data engineering use cases depend heavily on statistical models or machine learning. In the era of IoT, where thousands of devices and millions of sensors are inter-connected across distributed locations, there is a need to shift these applications to the edge. In large-scale IoT environments like a smart city, NES is going to enable us to push down aggregate or filter operations with on-the-fly analytical decisions. By executing these complex analytical tasks in early stages of data pipelines, we have knowledge what data to forward across the network and thus reduce network load and latency.

4 Conclusion

This paper presented challenges and opportunities for an IoT data management system to enable complex analytics beyond the cloud. To this end, we described changes to existing core features, investigated new features, as well as domain-specific features. For each feature, we presented the challenges and requirements, discussed the state-of-the-art, showed their limitations, and highlighted how we can use IoT infrastructures efficiently. Furthermore, we outlined our envisioned solutions for these challenges in our own system NebulaStream. We hope that this paper lays the foundation for a new type of systems that leverages the IoT to enable large-scale applications over millions of IoT devices in highly dynamic and geo-distributed environments. Overall, we envision NebulaStream and its new set of features and possibilities as a major step towards the smart city of the future.

References

[1] D. J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. R. Stonebraker, N. Tatbul, and S. B. Zdonik, "Aurora: a new model and architecture for data stream management," *VLDBJ*, 2007.

[2] L. Affetti, A. Margara, and G. Cugola, "Flowdb: Integrating stream processing and consistent state management," in *DEBS*, 2017.

[3] A. Ahmed, H. Arkian, D. Battulga, A. J. Fahs, M. Farhadi, D. Giouroukis, A. Gougeon, F. O. Gutierrez, G. Pierre, and P. R. Souza Jr, "Fog computing applications: Taxonomy and requirements," *arXiv:1907.11621*, 2019.

[4] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, and J. Saltz, "Hadoop-gis: A high performance spatial data warehousing system over mapreduce," in *PVLDB*, 2013.

[5] 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," *PVLDB*, 2013.

[6] Amazon, "Amazon aws greengrass," accessed December 15, 2019. [Online]. Available: https://aws.amazon.com/greengrass

[7] Amazon, "Aws iot analytics," retrieved December 15, 2019. [Online]. Available: https://aws.amazon.com/iot-analytics

[8] A. Arasu, B. Babcock, and S. Babu, "Stream: The stanford data stream management system," in *Data Stream Management*. Springer, 2016, pp. 317–336.

[9] C. Balkesen, N. Tatbul, and M. T. Özsu, "Adaptive input admission and management for parallel stream processing," in *DEBS*, 2013.

[10] D. Bandyopadhyay and J. Sen, "Internet of things: Applications and challenges in technology and standardization," *WPC*, vol. 58, 2011.

[11] L. Benson, P. M. Grulich, S. Zeuch, V. Markl, and T. Rabl, "Disco: Efficient distributed window aggregation," in *EDBT*. OpenProceedings.org, 2020, pp. 423–426.

[12] I. Botan, P. M. Fischer, D. Kossmann, and N. Tatbul, "Transactional stream processing," in *EDBT/ICDT*, 2012.

[13] S. Brenner, C. Wulf, D. Goltzsche, N. Weichbrodt, M. Lorenz, C. Fetzer, P. Pietzuch, and R. Kapitza, "Securekeeper: confidential zookeeper using intel sgx," in *Middleware*, 2016.

[14] L. Burkhalter, A. Hithnawi, A. Viand, H. Shafagh, and S. Ratnasamy, "Timecrypt: Encrypted data stream processing at scale with cryptographic access control," in *NSDI*, 2020.

[15] P. Carbone, S. Ewen, G. Fóra, S. Haridi, S. Richter, and K. Tzoumas, "State management in apache flink: Consistent stateful distributed stream processing," *PVLDB*, 2017.

[16] P. Carbone, G. E. Gévay, G. Hermann, A. Katsifodimos, J. Soto, V. Markl, and S. Haridi, "Large-scale data stream processing systems," in *Handbook of Big Data Technologies*. Springer, 2017.

[17] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *TCDE*, 2015.

[18] 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," *PVLDB*, 2014.

[19] J. Chen, L. Ramaswamy, D. K. Lowenthal, and S. Kalyanaraman, "Comet: Decentralized complex event detection in mobile delay tolerant networks," in *MDM*, 2012.

[20] G. Cugola and A. Margara, "Deployment strategies for distributed complex event processing," *Computing*, 2013.

[21] T. S. Darwish and K. A. Bakar, "Fog based intelligent transportation big data analytics in the internet of vehicles environment: motivations, architecture, challenges, and critical issues," *IEEE Access*, 2018.

[22] B. Del Monte, S. Zeuch, T. Rabl, and V. Markl, "Rhino: Efficient management of very large distributed state for stream processing engines," in *SIGMOD*, 2020.

[23] A. J. Demers, J. Gehrke, B. Panda, M. Riedewald, V. Sharma, and W. M. White, "Cayuga: A general purpose event monitoring system." in *CIDR*, 2007.

[24] B. Derakhshan, A. R. Mahdiraji, T. Rabl, and V. Markl, "Continuous deployment of machine learning pipelines." in *EDBT/ICDT*, 2019.

[25] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *TCC*, 2006.

[26] A. Eldawy, "SpatialHadoop: Towards flexible and scalable spatial processing using MapReduce," in *SIGMOD, PhD Symposium*, 2014.

[27] S. Ewen, "Data artisans streaming ledger serializable acid transactions on streaming data," 2018. [Online]. Available: https://www.ververica.com/blog/serializable-acid-transactions-on-streaming-data

[28] I. Flouris, N. Giatrakos, A. Deligiannakis, M. Garofalakis, M. Kamp, and M. Mock, "Issues in complex event processing: Status and prospects in the big data era," *JSS*, pp. 217–236, 2017.

[29] Z. Galić, E. Mešković, and D. Osmanović, "Distributed processing of big mobility data as spatio-temporal data streams," *Geoinformatica*, pp. 263–291, 2017.

[30] S. Gallagher. Ms researchers claim to crack encrypted database with old simple trick. https://arstechnica.com/information-technology/2015/09/ms-researchers-claim-to-crack-encrypted-database-with-old-simple-trick/.

[31] H. Gavriilidis, A. Michalke, L. Mons, S. Zeuch, and V. Markl, "Scaling a public transport monitoring system to internet of things infrastructures." in *EDBT/ICDT*, 2020, pp. 627–630.

[32] L. Girod, Y. Mei, and R. Newton, "The case for a signal-oriented data stream management system," in *CIDR*, 2007.

[33] P. M. Grulich and F. Nawab, "Collaborative edge and cloud neural networks for real-time video processing," *PVLDB*, 2018.

[34] P. M. Grulich, B. Sebastian, S. Zeuch, J. Traub, J. v. Bleichert, Z. Chen, T. Rabl, and V. Markl, "Grizzly: Efficient stream processing through adaptive query compilation," in *SIGMOD*, 2020, pp. 2487–2503.

[35] Y. He, S. Barman, D. Wang, and J. F. Naughton, "On the complexity of privacy-preserving complex event processing," in *PODS*, 2011, pp. 165–174.

[36] M. Hung, "Leading the iot, gartner insights on how to lead in a connected world," *Gartner Research*, 2017.

[37] J.-H. Hwang, M. Balazinska, A. Rasin, U. Cetintemel, M. Stonebraker, and S. Zdonik, "High-availability algorithms for distributed stream processing," in *ICDE*, 2005, pp. 779–790.

[38] A. Iyer, L. E. Li, and I. Stoica, "Celliq : Real-time cellular network analytics at scale," in *NSDI*, 2015.

[39] A. P. Iyer and I. Stoica, "A scalable distributed spatial index for the internet-of-things," in *SoCC*, 2017.

[40] G. Jia, G. Han, A. Li, and J. Du, "Ssl: Smart street lamp based on fog computing for smarter cities," *IEEE TII*, 2018.

[41] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ASPLOS*, pp. 615–629, 2017.

[42] J. Karimov, T. Rabl, and V. Markl, "Astream: Adhoc shared stream processing," in *SIGMOD*, 2019, pp. 607–622.

[43] N. R. Katsipoulakis, A. Labrinidis, and P. K. Chrysanthis, "A holistic view of stream partitioning costs," *PVLDB*, pp. 1286–1297, 2017.

[44] A. Kipf, H. Lang, V. Pandey, R. A. Persa, C. Anneser, E. Tzirita Zacharatou, H. Doraiswamy, P. A. Boncz, T. Neumann, and A. Kemper, "Adaptive main-memory indexing for high-performance point-polygon joins." in *EDBT/ICDT*, 2020, pp. 347–358.

[45] I. Kolchinsky and A. Schuster, "Join query optimization techniques for complex event processing applications," *PVLDB*, 2018.

[46] 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 *SIGMOD*, 2015, pp. 239–250.

[47] M. Liu, "Modeldb," 2020. [Online]. Available: https://github.com/VertaAI/modeldb

[48] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tinydb: an acquisitional query processing system for sensor networks," *TODS*, pp. 122–173, 2005.

[49] A. R. Mahmood, A. Daghistani, A. M. Aly, M. Tang, S. Basalamah, S. Prabhakar, and W. G. Aref, "Adaptive processing of spatial-keyword data over a distributed streaming cluster," in *SIGSPATIAL*, 2018, pp. 219–228.

[50] J. Meehan, C. Aslantas, S. Zdonik, N. Tatbul, and J. Du, "Data ingestion for the connected world." in *CIDR*, 2017.

[51] J. Meehan, N. Tatbul, S. Zdonik, C. Aslantas, U. Cetintemel, J. Du, T. Kraska, S. Madden, D. Maier, and A. Pavlo, "S-store: streaming meets transaction processing," *PVLDB*, pp. 2134–2145, 2015.

[52] H. Miao, H. Park, M. Jeon, G. Pekhimenko, K. S. McKinley, and F. X. Lin, "Streambox: Modern stream processing on a multicore machine," in *ATC*, 2017, pp. 617–629.

[53] MLflow, "An open source platform for the machine learning lifecycle," 2020. [Online]. Available: https://mlflow.org/

[54] M. Naehrig, K. Lauter, and V. Vaikuntanathan, "Can homomorphic encryption be practical?" in *CCSW*, 2011.

[55] M. Nikolic, B. Chandramouli, and J. Goldstein, "Enabling signal processing over data streams," in *SIGMOD*, 2017.

[56] D. O'Keeffe, T. Salonidis, and P. Pietzuch, "Frontier: Resilient edge processing for the internet of things," *PVLDB*, 2018.

[57] H. Park, S. Zhai, L. Lu, and F. X. Lin, "Streambox-tz: secure stream analytics at the edge with trustzone," in *ATC*, 2019.

[58] R. A. Popa, C. M. Redfield, N. Zeldovich, and H. Balakrishnan, "Cryptdb: protecting confidentiality with encrypted query processing," in *SOSP*, 2011, pp. 85–100.

[59] D. Reinsel, J. Gantz, and J. Rydning, "Data age 2025: The digitization of the world from edge to core," 2018, accessed December 15, 2019, from https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf.

[60] N. P. Schultz-Møller, M. Migliavacca, and P. Pietzuch, "Distributed complex event processing with query rewriting," in *DEBS*, 2009, pp. 1–12.

[61] C. Segarra, R. Delgado-Gonzalo, M. Lemay, P.-L. Aublin, P. Pietzuch, and V. Schiavoni, "Using trusted execution environments for secure stream processing of medical data - (case study paper)," in *DisCoTec*, 2019.

[62] Z. Shen, V. Kumaran, M. J. Franklin, S. Krishnamurthy, A. Bhat, M. Kumar, R. Lerche, and K. Macpherson, "Csa: Streaming engine for internet of things." in *IEEE*, 2015, pp. 39–50.

[63] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE IoT Journal*, 2016.

[64] ShotSpotter, https://www.shotspotter.com/, accessed 25th March 2020.

[65] D. Sidlauskas, S. Chester, E. Tzirita Zacharatou, and A. Ailamaki, "Improving spatial data processing by clipping minimum bounding boxes," in *ICDE*, 2018, pp. 425–436.

[66] L. Sweeney, "K-anonymity: A model for protecting privacy," *IJUFKS*, 2002.

[67] M. Tang, Y. Yu, Q. M. Malluhi, M. Ouzzani, and W. G. Aref, "Locationspark: A distributed in-memory data management system for big spatial data," *PVLDB*, pp. 1565–1568, 2016.

[68] A. Toshniwal, S. Taneja, and A. Shukla, "Storm@ twitter," in *SIGMOD*, 2014.

[69] E. Tzirita Zacharatou, H. Doraiswamy, and et al., "GPU Rasterization for Real-Time Spatial Aggregation over Arbitrary Polygons," *PVLDB*, 2017.

[70] M. Vuppalapati, J. Miron, and R. Agarwal, "Building an elastic query engine on disaggregated storage," in *NSDI*, 2020.

[71] D. Wang, E. A. Rundensteiner, and R. T. Ellison III, "Active complex event processing over event streams," *PVLDB*, pp. 634–645, 2011.

[72] D. Xie, F. Li, B. Yao, G. Li, L. Zhou, and M. Guo, "Simba: Efficient in-memory spatial analytics," in *Proceedings of the 2016 International Conference on Management of Data*, 2016, pp. 1071–1085.

[73] W. Yao, C. H. Chu, and Z. Li, "Leveraging complex event processing for smart hospitals using rfid," *JNCA*, 2011.

[74] Y. Yao and J. Gehrke, "The cougar approach to in-network query processing in sensor networks," in *SIGMOD*, 2002.

[75] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," in *MBD*, 2015.

[76] J. Yu, J. Wu, and M. Sarwat, "Geospark: A cluster computing framework for processing large-scale spatial data," in *SIGSPATIAL*, 2015.

[77] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica, "Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters," in *HotCloud*, 2012.

[78] S. Zeuch, A. Chaudhary, B. Monte, H. Gavriilidis, D. Giouroukis, P. Grulich, S. Breß, J. Traub, and V. Markl, "The nebulastream platform: Data and application management for the internet of things," in *CIDR*, 2020.

[79] S. Zeuch, B. D. Monte, J. Karimov, C. Lutz, M. Renz, J. Traub, S. Breß, T. Rabl, and V. Markl, "Analyzing efficient stream processing on modern hardware," *PVLDB*, 2019.

[80] H. Zhang, Y. Diao, and N. Immerman, "On complexity and optimization of expensive queries in complex event processing," in *SIGMOD*, 2014.

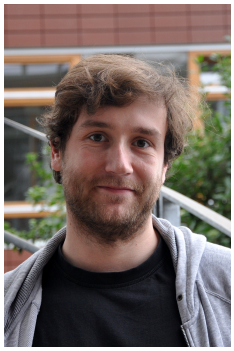
[81] S. Zhang, J. He, and A. C. Zhou, "Briskstream: Scaling data stream processing on shared-memory multicore architectures," in *SIGMOD*, 2019.

[82] S. Zhang, H. T. Vo, and D. Dahlmeier, "Multi-query optimization for complex event processing in sap esp," in *ICDE*, 2017.

[83] S. Zhang, Y. Wu, F. Zhang, and B. He, "Towards concurrent stateful stream processing on multicore processors," in *ICDE*, 2020, pp. 1537–1548.

[84] B. Zhao, N. Q. V. Hung, and M. Weidlich, "Load shedding for complex event processing: Input-based and state-based techniques," in *ICDE*, 2020.

AUTHOR BIOGRAPHIES



Steffen Zeuch is a Senior Researcher at the DIMA group (TU Berlin) and IAM group (DFKI). He received his Ph.D. in Computer Science at Humboldt University Berlin in the research group of Prof. Freytag. Steffen is conducting research in data management, with an emphasis on topics related to modern hardware, distributed systems, and IoT environments. Currently, he is the project lead of the NebulaStream (www.nebula.stream) project at DIMA, which builds a new data management for the Internet of Things. He has published research papers on query optimization and execution as well as on novel system architectures in many top-tier conferences.

Eleni Tzirita Zacharatou is a Senior Researcher at the DIMA group (TU Berlin). Her current research interests are centered around robust stream processing in mobile IoT environments. Eleni received her PhD from the École Polytechnique Fédérale de Lausanne (EPFL) in 2019, where she worked in the DIAS lab on spatio-temporal query processing algorithms and indexing methods for data exploration. She holds a Diploma - M.Eng. degree in Electrical and Computer Engineering from the National Technical University of Athens. Eleni is a recipient of the 2018 ACM SIGMOD best demonstration award.

Shuhao Zhang is a Senior Researcher at the DIMA group (TU Berlin). Shuhao's research interests include modern hardware and stream processing. He has published research papers on stream processing system design and optimization on novel hardware architectures. He did his Ph.D. in Computer Science at National University of Singapore.

Xenofon Chatziliadis is a Ph.D. candidate at DIMA group (TU Berlin). Xenofons's research interests include distributed systems, performance monitoring, and IoT environments. He received his M.Sc. in Computer Science at TU Berlin with a focus on machine learning.

Ankit Chaudhary is a Ph.D. candidate at DIMA group (TU Berlin). Ankit's research interests include distributed systems, query optimization, and IoT environments. He did his M.Sc. from TU Kaiserslautern with focus on distributed systems and B.Tech. in information and technology from GGSIPU Delhi.

Bonaventura Del Monte is a Ph.D. candidate at DIMA group (TU Berlin). Bonaventura's research interests include efficient query execution on modern hardware, stateful stream processing, and data-intensive distributed systems. He got his M.Sc. in Computer Science from University of Salerno.

Dimitrios Giouroukis is a Ph.D. candidate at DIMA group (TU Berlin). Dimitrios' research interests include sensor data management, stream processing, and post-cloud distributed systems. He got his M.Sc. in Computer Science from Aristotle University of Thessaloniki.

Philipp M. Grulich is a Ph.D. candidate at DIMA group (TU Berlin). Philipp's research interests include modern hardware, stream processing, and compiler theory. He received his M.Sc. in Computer Science at TU Berlin with a focus on query compilation.

Ariane Ziehn is a Ph.D. candidate at the IAM group (DFKI). Ariane's research interests include distributed systems, complex event processing, and IoT environments. She received her M.Sc. in Information Systems Management at TU Berlin with a focus on distributed systems as well as data and software engineering.

Volker Markl is a Full Professor and Chair of the Database Systems and Information Management (DIMA) Group at the Technische Universität Berlin (TU Berlin). At the German Research Center for Artificial Intelligence (DFKI), he is Chief Scientist and Head of the Intelligent Analytics for Massive Data Research Group. In addition, he is Director of the Berlin Institute for the Foundations of Learnig and Data (BIFOLD), a merger of the Berlin Big Data Center (BBDC) and the Berlin Center for Machine Learning (BZML). Volker Markl is a computer science graduate from Technische Universität München, where he earned his Diplom in 1995 with a thesis on exception handling in programming languages. He earned his PhD in 1999 the area of multidimensional indexing under the supervision of Rudolf Bayer. Volker Markl has published numerous research papers on indexing, query optimization, lightweight information integration, and scalable data processing. He holds 18 patents, has transferred technology into several commercial products, and advises several companies and startups.