The NebulaStream Platform: Data and Application Management for the Internet of Things

Steffen Zeuch[1,2] Ankit Chaudhary[1] Bonaventura Del Monte[1,2] Haralampos Gavriilidis[1]
Dimitrios Giouroukis[1] Philipp M. Grulich[1] Sebastian Breß[1] Jonas Traub[1,2] Volker Markl[1,2]

[1]Technische Universität Berlin [2]DFKI GmbH

ABSTRACT

The Internet of Things (IoT) presents a novel computing architecture for data management: a distributed, highly dynamic, and heterogeneous environment of massive scale. Applications for the IoT introduce new challenges for integrating the concepts of fog and cloud computing in one unified environment. In this paper, we highlight these major challenges and showcase how existing systems handle them. Based on these challenges, we introduce the NebulaStream platform, a general purpose, end-to-end data management system for the IoT. NebulaStream addresses the 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 our evaluation, we demonstrate the effectiveness of our approach by providing early results on partial aspects.

1. INTRODUCTION

Over the last decade, the amount of produced data has reached unseen magnitudes. Recently, the International Data Corporation [15] estimated that by 2025 the global amount of data will reach 175ZB and that 30% of this data will be gathered in real-time. In particular, the number of IoT devices increases exponentially such that the IoT is expected to grow as large as 20 billion connected devices in 2025 [6]. At the same time, devices such as embedded computers or mobile phones increase their processing capabilities continuously. This trend enables the exploitation of their computing and communication capabilities as they become objects of common use. As a result, the IoT is one of the fastest emerging trends in the area of information and communication technology [11]. The explosion in the number of devices will create novel data-driven applications in the near future. These applications require low-latency, location awareness, wide-spread geographical distribution, and real-time data processing on potentially millions of distributed data sources. To enable these applications, a data management system needs to leverage the capabilities of the IoT devices.

However, today's classical data management systems are not ready yet for these applications as they embrace either the cloud or the fog computing paradigm. Systems based on the cloud paradigm, e.g., Flink, Spark, or Kafka Streams, do not yet exploit the full capabilities of IoT devices. To implement an IoT application using current systems, the user needs to collect sensor data centrally in a cluster before he can apply processing. However, upcoming IoT applications require the processing of data from millions of distributed sensors, which would have to be sent to a single server; thus, introducing a severe bottleneck. In contrast, systems based on the fog computing paradigm, e.g., Frontier [13], CSA [17], or TinyDB [10] exploit the processing capabilities of edge devices, i.e., devices that are physically closer to the end-users. These devices enable data reduction techniques, e.g., pre-selection or pre-aggregation, to reduce the data volume as early as possible in the processing pipeline, i.e., close to the sensor. However, fog computing systems only scale within the fog and do not exploit the virtually unlimited resources of modern cloud infrastructures. Finally, there is no fog-based, general purpose, end-to-end data management system with functionality similar to cloud-based systems such as Flink or Spark.

To enable future IoT applications, a system has to combine the cloud, the fog, and the sensors in a single unified platform to leverage their individual advantages and exploit cross-paradigm optimizations. From a system point of view, this unified environment imposes three unique characteristics that are not supported by state-of-the-art data processing systems.

Heterogeneity: A unified environment consists of a highly heterogeneous hardware landscape. The processing nodes range from low-end system-on-a-chip devices (e.g., Raspberry PIs) to high-end rack-scale servers. In particular, cloud infrastructures consist of homogeneous node setups, whereas the fog contains heterogeneous, low-end computing devices.

Unreliably: A unified environment has to handle different levels of runtime dynamics. The fog introduces a highly dynamic runtime environment with unreliable fog nodes that might change their geospatial position, i.e., resulting in many transient errors or changes in latency and throughput. In contrast, a cloud infrastructure is a relatively stable environment where node failures are seldom.

Elasticity: In a unified environment, data move from the sensors via intermediate nodes to the cloud, and finally to the consumer. The Fog Layer is built as a tree-like network topology [3, 5] with several data-flow paths. Data processing in the Fog Layer has to be network-aware because only nodes on the path from the sensors to the cloud can participate. In contrast, in the cloud, every node can potentially access all data, e.g., via a distributed file system.

Overall, a unified environment introduces a previously unprecedented, unique combination of characteristics, i.e., hardware heterogeneity, unreliable nodes, and changing network topologies. In this paper, we propose *NebulaStream* (NES), a novel data processing platform that addresses the above mentioned heterogeneity, dynamicity, distribution, and scalability

9th Biennial Conference on Innovative Data Systems Research (CIDR '19) January 13-16, 2019 , Asilomar, California, USA.

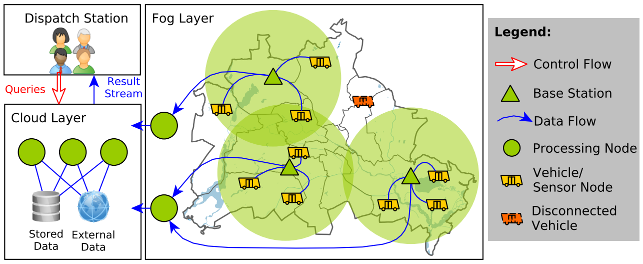

Figure 1: IoT Application Scenario.

challenges and enables effective and efficient data management for the IoT. In NES, we cope with a heterogeneous environment by maximizing *sharing of results* and *efficiency of computing* to significantly reduce the amount of data transferred and to exploit hardware capabilities efficiently. We cope with an unreliable environment in NES by applying *dynamic decisions* and *incremental optimizations* during runtime to be as flexible as possible. We cope with an elastic environment in NES by designing each node to react autonomously to a wide range of situations during runtime.

With NES, we enable future IoT applications by unifying sensors, fog, and cloud in one general purpose, end-to-end data management platform. Our early experiments show that NES reduces the amount of data and sensor reads up to 90%, increases node throughput on low-end devices by up to two orders of magnitude, and process with low latency even in the presence of many node failures.

The remainder of the paper is structured as follows. We show a typical IoT application scenario in Section 2. In Section 3, we describe the NebulaStream platform, discuss its design principles, and provide initial performance results. After that, we survey related work in Section 4 and conclude in Section 5.

2. IoT APPLICATION SCENARIO

In Figure 1, we present an integrated public transport system of Berlin as a representative IoT application scenario. The components in this scenario are either stationary or mobile. Vehicles (red and yellow boxes), i.e., taxis, buses, subways, and trains move around and carry a set of sensors and a simple processing unit. Each unit collects vehicle data (e.g., routing, maintenance information, and occupancy/usage) as well as data from the environment (e.g., traffic, road conditions, and weather). The base stations, processing nodes, and dispatch station represent stationary components. Base Stations (green triangles) are distributed across the city and have antennas, network routers, and compute and storage capacity. Processing nodes (green circles) are distributed within the city to gather data from several base stations and apply more complex processing. The centralized dispatch station represents the endpoint for all data. This station collects data from the fog and the cloud to enable the authority to manage the public transport. This IoT application scenario requires a massively distributed system with continuous data producers as well as transient and permanent, distributed compute and storage capabilities. The environment in this scenario differs fundamentally from current cloud architectures. In particular, vehicles move within the city and interact with multiple antennas, which transmit data to base stations. Due to the dynamic nature, vehicles may encounter temporary connection losses or outages (red vehicle), e.g., they are outside of transmission ranges. Furthermore, all vehicles move at different speeds, on different roads/tracks, and are potentially equipped with different hardware. As a result, a fog requires continuous adaptation to a dynamic environment with respect to faults and changes in the availability, amount, type, capacity, and location of data and compute nodes. Despite the distributed nature, it must be possible to manage the system with a centralized, global view and execute continuous as well as ad-hoc data analytics. This includes the entire data analysis pipeline, from information extraction to integration and model building using machine learning, signal processing, and other advanced analytics.

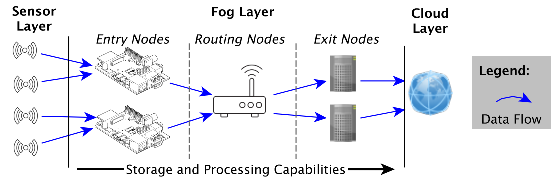

Figure 2: An NES Topology.

From a user perspective, this system may assist the dispatcher to schedule new vehicles or re-route vehicles in case of outages or increased passenger demands. This results in a feedback loop that may change the physical fog architecture. Furthermore, this architecture allows for enriching real-time data with external sources, e.g., air pollution measurements, event calendars, area crowdedness, or knowledge bases. The characteristics of this application are representative of many IoT scenarios including Industry 4.0, smart homes, smart grids, smart cities, or participatory sensing applications.

3. NebulaStream PLATFORM (NES)

In this section, we present the NebulaStream (NES) platform. First, we investigate the common topology of IoT scenarios and highlight their novelty (Section 3.1). After that, we identify key design principles for an IoT platform (Section 3.2) and later describe how NES implements them (Section 3.3). Finally, we discuss challenges that an IoT system has to tackle and our solutions for them in NES (Section 3.4).

3.1 NES Topology

In Figure 2, we present the multi-layer fog topology that is common in today's IoT infrastructure [3]. This figure presents the data-flow from the sensors to the cloud. The basic assumptions in this topology are two-fold. First, all data might reach the *Cloud Layer*. Second, the Cloud Layer is able to apply remaining processing, i.e., representing a fall-back mechanism. In contrast, all other nodes can only access data if they are routed through them and their storage and processing capabilities determine the operations they can apply.

On the *Sensor Layer*, millions of sensors produce data without processing them. Sensors provide two data access patterns: pull-based and push-based. Each sensor is connected to at least one low-end node in the *Fog Layer*, which is responsible for this sensor (so-called *Entry Node*). In the *Fog Layer*, NES processes data as they flow from Entry Nodes to Exit Nodes and while nodes may change their geospatial position. The routing is performed by *Routing Nodes* such as routers or switches. The data processing capabilities on Routing Nodes are restricted and the provided functionality is highly vendor-dependent [2]. In general, the storage and processing capabilities of nodes increase significantly in the NES Topology with each hop towards the Cloud Layer. After leaving the Fog Layer through an *Exit Node*, data enter the Cloud Layer. The Cloud Layer provides virtually unlimited scaling of processing and storage. In IoT application scenarios, this layer will perform the remaining computation and output the data to the user.

The NES Topology introduced in Figure 2 represents a fundamentally new and unique set of characteristics and requirements compared to common cloud infrastructures. First, the query processing and the operator placement have to be network-aware. The main query optimization goal is to find an efficient route

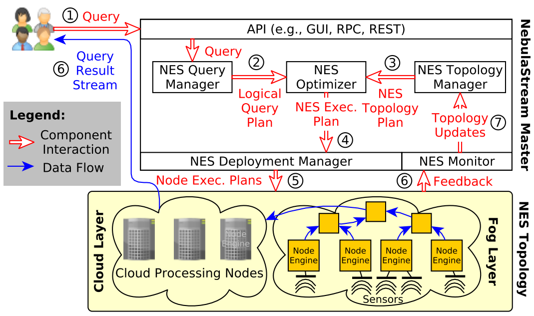

Figure 3: Architecture Overview.

through the Fog Layer that reduces data volumes as early as possible without violating any SLA but fulfilling the Quality of Service (QoS) constraints. Second, the NES Topology is highly heterogeneous and many nodes have only limited computation capabilities. In particular, nodes in the lower parts of the Fog Layer are restricted in storage and computational capabilities. Furthermore, processing has to trade-off between power and performance. Third, the Fog Layer is highly unreliable compared to the homogeneous and relatively stable Cloud Layer. Fourth, the volume and velocity of sensor data represent an external factor that cannot be controlled. As a result, the entire system has to evolve around sensor data that is injected by the environment. With NES, we build a platform that creates a federation of sensors, fog, and cloud, which enables big data acquisition and analysis.

3.2 NES Design Principles

With NES, we build a platform for future IoT applications that copes with the unique set of characteristics of a unified environment. For individual layers, different approaches were proposed over the last decades. However, combining all of them into a single system is the major challenge that we address with NES. To handle millions of sensors and thousands of queries, we base the system design of NES on the following design principles:

1. **Dynamic Decisions:** NebulaStream never expects a static behavior or conditions in any component. We equip nodes with all logic necessary to act as autonomously as possible.
2. **Incremental Optimizations:** NebulaStream optimizes a network of active queries in incremental steps rather than traditional query optimization or batched plan changes.
3. **Maximize Sharing:** NebulaStream shares data and processing wherever possible, i.e., on windows (stream slicing), among queries (multi-query optimization), on sensor data, and on operator level (code optimization).
4. **Maximize Efficiency:** NebulaStream exploits the underlying hardware most efficiently by applying hardware-tailored code generation and optimizations. This enables extreme scalability to process data on millions of small scale devices.
5. **SLA Centric Processing:** NebulaStream's primary goal is to match user-provided SLAs and QoS constraints with the available resources.
6. **Ease of Use:** NebulaStream enables users to choose their preferred programming environments and models, without worrying about system-internals and performance implications.

3.3 NES Architecture

In Figure 3, we present the architecture of NebulaStream. In general, NES implements a client/server architecture. Users interact with NES through one of the provided APIs to send queries to the *NES Master Node* (1). Our current API allows for specifying dataflow programs, similar to the APIs of streaming systems like Flink, Spark, and Storm. Inside the NES Master Node, several components orchestrate the query processing. The

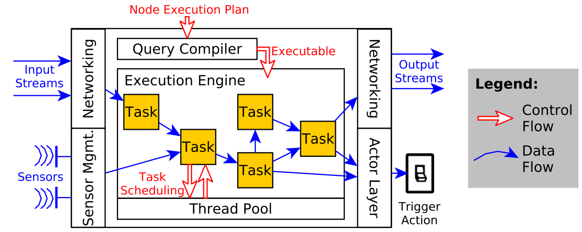

Figure 4: Node Engine.

NES Query Manager is responsible for creating logical query plans from user requests (2). Additionally, this component maintains *logical streams* that represent logical views over sensors, e.g., a logical stream *cars* could combine sensor inputs from multiple cars into one consistent stream. The *NES Topology Manager* orchestrates the NES Topology, which consists of workers and sensors. During startup, each device registers itself and provides information, such as resource capabilities and network topology information. The *NES Optimizer* provides the assignment of a logical query plan (created by the NES Query Manager) to the current NES Topology plan (3) (maintained by the NES Topology Manger); thus creating a *NES Execution Plan (NES-EP)*. The actual assignment introduces a large optimization space, e.g., operators can be assigned top-down, bottom-up, or using other assignment algorithms. The *NES Deployment Manager* takes the NES-EP (4), disassembles it into Node Execution Plans (Node-EPs), and deploys them to the nodes in the NES Topology, i.e., into either the Fog or the Cloud Layer (5). This deployment is performed incrementally and requires the addition of new operators on nodes as well as the rerouting of data on different data-flow paths. The *NES Monitor* constantly collects feedback from the NES Topology (6) and maintains statistics and current resource utilization for the NES Topology Manager (7).

In Figure 4, we show the components of the node engine, which is deployed on all devices of the NES Topology. The *Node Engine* is responsible for communicating with the NES Master Node, accepting Node-EPs and control messages as well as setting up the input sources, output sinks, and other components. The incoming queries are Node-EPs, which contain a partial subtree of the overall NES-EP. The Node-EP is compiled by the local query compiler and later injected into the processing tasks. As input, the Node Engine receives data from the network, e.g., from another node, or directly from an attached sensor. As output, the Node Engine either sends data over the network or triggers an action on an attached device, e.g., controlling an actuator such as a light switch. The *Execution Engine* orchestrates the processing inside each Node Engine. The central unit of work is one task that combines n input buffers, m output buffers, and the application of the specified data processing algorithm. The processing in NES is *source-driven* and applies the following sequence of steps on each incoming buffer. First, the engine assembles the tasks by embedding the executable and allocates all required input, intermediate, and output buffers. After that, the engine enqueues the tasks in one of the processing queues. Finally, each thread in the *Thread Pool* dequeues one task, processes it, and either enqueues the result buffer into an output queue or triggers an action. This highly dynamic design enables high resource utilization but also introduces a dynamic execution order, which poses new challenges for the system design.

In addition to processing components, each node engine contains dedicated components for windows, routing, sensors, and state. As a result, we drastically reduce the complexity of the query compiler and increase maintainability and separation of concerns in NES. In particular, NES compiles only the *hot* code fragments and link other functionalities as pre-compiled components (following Neumann et al. [12]).

Overall, it is a design decision in NES to equip the Node Engine with all necessary components to enable it to be as autonomous as possible. In particular, we assign all means to the node to enable it to make as many decisions as possible decentrally and independently without reaching out to a central component. With this design, we address the transient connection losses within the unreliable fog topology. Furthermore, IoT applications potentially introduce a hierarchy of master nodes e.g., regional masters per street or per urban district. In NES, we envision a system design with autonomous nodes and a simple master to mitigate potential bottlenecks in large scale environments. Note that, we plan to investigate the trade-off between centralized and decentralized decisions in the future.

3.4 NES Solutions for IoT Challenges

Based on the unique characteristics highlighted in Section 1 and IoT application scenarios presented in Section 2, we outline five main challenges for an IoT data management system. In the following, we discuss the challenges and propose our solutions.

C1 - Heterogeneity, Distribution, and Volume of Data At-Rest and Data In-Motion: NebulaStream scales to thousands of queries and millions of sensors. A particular challenge originates from handling the sheer amount, potentially millions, of diverse data sources. These sources differ in their characteristics, ranging from millions of small sensor streams to a few large streams from sources such as click-streams or auctions. Furthermore, data sources either provide data in-motion or at-rest. The accessibility of sources under security and privacy constraints, as well as efficient access paths, requires solutions completely different from what today's big data processing systems provide. These characteristics imply research questions with respect to scalability, efficiency, integration, and interoperability. To support this high diversity in NES, we follow the *Maximize Sharing* design principle (Section 3.2) and apply data sharing techniques on three different levels. First, on the query level, NES exploits data sharing among multiple streaming queries as proposed by Karimov et al. [7]. Second, on the operator level, NES slices data streams and exploits data sharing on stream aggregations as proposed by Traub et al. [19]. Third, on the sensor level, NES applies *Acquisitional Query Processing* (ACQP) [10] and On-Demand scheduling of sensor reads and data transmissions [18] to limit data acquisition to the data points required for answering user queries.

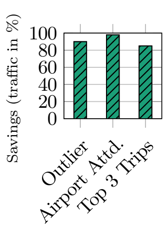

Figure 5: NES Data Reduction.

Figure 5 presents an initial experiment that demonstrates the potential savings of data reduction techniques in NES. We use the New York taxis data set, derive routes for each taxi trip, and replay the routes of all taxis on Raspberry Pis. We modify the data acquisition process for three queries such that only required data is sampled and transmitted. Our queries include outlier detection (e.g., taxis leaving their district), airport attendance monitoring, and a top 3 query for the longest ongoing trips. As shown in Figure 5, saved data traffic is significant for all queries.

By combining the introduced techniques in NES, we drastically reduce the amount of acquired, transferred, and processed data and thus enable IoT applications with thousands of queries over millions of sensors.

C2 - Heterogeneity, Distribution, and Volume of Compute: NebulaStream exploits the hardware resources of millions of heterogeneous devices efficiently. A particular challenge originates from a fog topology that consists of potentially hundreds of millions of compute devices with diverse capabilities with respect to storage, processing, and interconnect. The devices range from small battery-powered sensors with no compute capabilities (beyond simple filtering) and an unreliable temporary connection, to a large compute cluster with huge storage, infiniband interconnect, and thousands of compute cores. These characteristics imply challenges with respect to security, permission management, and efficient and effective resource utilization.

To support this heterogeneity in NES, we follow the *Maximize Efficiency* design principle (Section 3.2). In particular, we apply two techniques. First, we use query compilation as the leading paradigm for achieving high resource utilization in data-at-rest processing [12]. In NES, we transfer this approach to the special semantics of the fog and stream processing. In particular, NES generates specialized code depending on the actual query, hardware, and data characteristics [22]. Furthermore, NES distributes query optimization and code generation between the central master and the local node engine. On the master, NES performs global query optimizations (e.g., operator reorder) and splits the query into segments for individual devices. On the node engine, the query compiler produces hardware-tailored code to exploit the availability capabilities most efficiently.

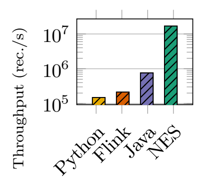

Figure 6: YSB on RaspberryPi 3B+.

Our experiment in Figure 6 compares the throughput of the Yahoo Streaming Benchmark (YSB) on a RaspberryPi 3B+ using Python, Flink, a hand optimized Java program, and NES. The results show that hardware-tailored code generation is essential to efficiently utilize resources, especially for low-end devices. Furthermore, through code generation, NES reduces the energy consumption per device and thus requires less computation to achieve the same performance. The trade-off between power consumption and performance is one major research area for NES in the future.

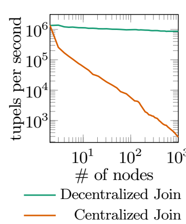

Figure 7: Join Exp.

As a second technique, we utilize in-network processing inside the Fog Layer to reduce the computation required at the Cloud Layer. In Figure 7, we present an example query that gathers values from up to 1k nodes and joins them to concise snapshots. In a centralized join, the Cloud Layer receives all data and performs a join based on the timestamp of incoming tuples. In a decentralized join, nodes exchange and join tuples incrementally. As shown in Figure 7, the centralized join causes a drastic throughput decay when the number of nodes increases. In contrast, by utilizing the available computing resources on the path from the sensors to the Cloud Layer, NES achieves almost constant throughput.

By applying specialized code generation and in-network processing, NES exploits the available compute resource most efficiently and allows to balance computational demands and energy consumption.

C3 - Spontaneous, Potentially Unreliable Connectivity between Data and Compute: NebulaStream detects and compensates potentially unreliable nodes in the Fog Layer without impacting consistency and availability. A particular challenge originates from the need to manage data and compute together, as most applications will consist of ad-hoc or standing queries as in streaming systems. Some data or compute units may be connected via Wifi, mobile, or satellite networks with intermittent connectivity and unreliable connections. In contrast to a homogeneous and relatively stable cloud environment, a heterogeneous and volatile fog environment has to handle frequent transient failures.

Failures in the fog occur due to numerous reasons, most notably hardware errors, software errors, congestion that results in back-pressure (straggler nodes), inadequate resource allocation, and transient connection lost. Furthermore, devices continuously refresh their connections while moving and create ad-hoc connections that result in an unpredictable communication pattern [11]. This requires special solutions that deal with the intermittent availability of resources, both with respect to data and code management. The resulting challenges address adaptivity, synchronization across devices, consistency/transaction management, recovery, and fault-tolerance.

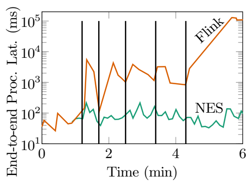

Figure 8: Fault Tolerance.

Current cloud-centric processing systems handle node failures using a stop-the-world approach. Thus, if an error occurs, the system stops the entire processing and redeploys a new query plan. In Figure 8, we show the end-to-end processing latency of Flink and NES while randomly terminating compute nodes (indicated by the black vertical lines). As shown, Flink cannot recover from high transient error rates as the latency constantly increases, whereas NES selectively restarts failed operators without halting the entire query.

To achieve reliability in an unreliable environment, we apply the *Dynamic Decisions* and *Incremental Optimizations* design principles in NES. We argue that a central component cannot keep up with the pace of failures in a dynamic, unified environment. On each level of the NES Topology, we apply different failure recovery approaches and thus provide different guarantees. On the Sensor Layer, NES substitutes missing sensor values from broken sensors with nearby sensors, if applicable, or buffers the values during transient connection loss [10]. On the Fog Layer, NES leverages the Frontier approach [13], which sends data through multiple network paths to achieve fault-tolerance. On the Cloud Layer, NES uses existing fault tolerance approaches, such as, global checkpointing and message brokers.

By extending and combining existing approaches on different levels of the NES Topology into a unified fault-tolerant solution, we enable the handling of spontaneous, potentially unreliable connectivity in the fog topology.

C4 - Diversity in Programming and Management Environments: NebulaStream supports a very diverse set of data processing workloads specified in different query languages and following different processing models (e.g., relational, linear, or graph algebra). A particular challenge originates from IoT applications that require a combination of different data-oriented programming paradigms. Possible workloads range over the entire data management pipeline, from information extraction over information integration to model building and model application. In particular, running AI/ML/Data Science algorithms in the fog enables direct feedback loops between the digital and the physical world. These workloads include potentially iterative algorithms mixing relational, linear, and graph algebra, and may run on top of continuous data streams or finite data sets. This diversity presents challenges with respect to 1) holistic, optimizable, intermediate representations, 2) efficient and scalable physical operators across all paradigms that can be mixed and matched, and 3) the combination of domain-specific and generic query languages that offers a sufficiently powerful yet optimizable interface to a data engineer. Furthermore, the programming and reasoning about sensors and actuators in such a distributed, diverse setting entails a huge challenge with respect to both, scalability and ease of use.

To support diverse workloads in NES, we apply the *Ease of Use* design principle. In NES, we allow users to choose their preferred programming environments and models and without taking system-internals and performance implications into account. To enable this diversity, we build on top of existing frameworks, such as Weld [14], Arc [8], Emma [1], and LARA [9] to represent diverse queries in a unified intermediate representation, the *Nebular-IR*. The Nebular-IR allows us to perform optimizations across operators, processing models, and language boundaries. The optimizations range from high-level optimizations on the operator plan level (e.g., placement, ordering, fusion [4]) down to low-level optimizations on the instruction level (e.g., branch conversion across operators).

From a management point of view, centrally managing the system in a heterogeneous distributed setup introduces challenges in areas such as data collection, response time, and fault-tolerance. To this end, NES provides a management view with a centralized, homogeneous interface, a automatic distribution, parallelization, and means to adaptively detect and react to changes in the environment.

By providing a central management view as well as a intermediate representation in NES, we support a diverse set of data processing workloads specified in different query languages and following different processing models.

C5 - Constant Evolution under Continuous Operation: NebulaStream supports continuous operations while the NES Topology and user workloads change constantly. A particular challenge originates from a changing topology where new devices join the fog, existing devices get phased out, or change their geospatial position. Additionally, the workloads continuously change as users submit, update, or delete queries. Furthermore, to enable time-sensitive processing, nodes must behave dynamically and autonomously during runtime, to capture and react to changes in velocity, volume, and variety. Managing and reacting to changes in a robust way while the system is in continuous operation presents drastic challenges to the software architecture and fabric of an IoT data management system.

To support such a highly dynamic environment in NES, we apply the *Dynamic Decisions* and *Incremental Optimizations* design principles. First, we design NES to equip the nodes with all necessary components to autonomously react to a wide range of situations. We enrich the Node-EPs with several alternative routes and different options. As a result, if a node detects changes in velocity, volume, or variety, it reacts dynamically at runtime. Second, we apply incremental optimizations such that NES modifies a stateful execution plan of a running query in incremental steps rather than by one large change. With each incoming or modified query as well as with each change in data velocity, volume, or variety, NES converges to the optimal NES-EP. Furthermore, we introduce continuous feedback loops between the NES Master Node and NES Node Engines in different layers to enable a central management in a heterogeneous distributed setup. The trade-off between centralized control in a master node and decentralized decisions in the nodes remains an open research field for the future.

By defining feedback loops between its components and by performing changes incrementally and autonomously, NES becomes resilient against constantly changing user workloads and network topologies.

Overall, NES addresses all challenges of an IoT system presented in Section 3.4 by combining existing approaches with new solutions. To this end, NES handles heterogeneous and distributed data sources and formats, utilizes available resources efficiently, copes with unstable network topologies, and provides multiple query and processing models.

4. STATE-OF-THE-ART SYSTEMS

In this section, we summarize current approaches and examine how they address the challenges of an IoT system (Section 1). Note that, the long version of this paper will cover related work in more detail.

Cloud-centric IoT data processing The first group of approaches relies on the cloud to process IoT data centrally. Mobile Cloud Computing (MCC) outsources data storage and processing from devices to the cloud. In this scenario, a pool of sensors gather and send data directly to a cloud infrastructure for further processing. As soon as data reach the cloud, common SPEs, such as Apache Flink or Apache Pulsar, process the incoming streams. Based on this infrastructure, cloud providers offer services to deploy and manage these data streams.

The cloud-centric processing of sensor data enables elastic scaling of compute and storage resources once data reach the cloud. However, this neglects the resources provided by sensors and intermediate nodes (**C1**,**C2**). Although these systems offer fault-tolerance and dynamic scaling (addressing **C3**,**C5**) in the cloud, they do not provide them across a unified sensor-fog-cloud environment. In NES, we extend existing work in the area of stream processing to incorporate IoT specific requirements.

Edge-aware IoT data processing. With the concept of Mobile-Edge Computing (MEC), cloud providers address the limitations of cloud-centric approaches by implementing *hub devices* to extend their IoT services [17]. Hub devices are placed at the edge of the fog topology and act as local control centers close to the sensors. They gather data from attached sensors, perform simple processing steps, and do not require a stable connection to a cloud infrastructure.

Although MEC and MCC improve scalability with respect to the number of attached sensors (addressing **C1**), they do not focus on efficient resource utilization across heterogeneous devices (**C2**). Furthermore, these approaches offer fault-tolerance only between hub-devices and the cloud but still require a stable connection between sensors and the hub-device (partially addressing **C3**). Additionally, these approaches do not address dynamic changes in the topology (**C3**).

Ryden et al. [16] introduced a distributed data and resource management framework. They leverage distributed in-situ data and computing resources on edge nodes for *batch processing*. Their system supports the combination of dedicated and voluntary resources under a unified infrastructure while ensuring high availability (addressing **C5**, partially addressing **C1**). However, their framework neither exploits hardware heterogeneity for efficient code computation nor supports a multi programming environment (**C2, C4**). In NES, we leverage this work to support a unified sensor-fog-cloud environment.

Fog-aware IoT data processing. Two data processing systems utilize the fog as the underlying infrastructure. O'Keeffe et al. [13] propose Frontier, a distributed and resilient data processing system for fog devices. Frontier aims to handle a large number of sensors and to achieve reliability. To this end, it exploits the processing capability of the fog by distributing queries over a topology (addressing **C1**). It replicates operators to neighboring nodes to recompute intermediate results and to cope with device failures (addressing **C3**). However, Frontier does not address the efficient utilization of heterogeneous devices, diversity in programming environments, and adaptability to the constant evolution of the fog (**C2**,**C4**,**C5**). Finally, it does not consider the exploitation of the cloud.

Zhitao et al. [17] extend Cisco's Connected Streaming Analytics platform (CSA) for IoT processing. CSA utilizes Cisco network hardware to enable in-network processing (partially addressing **C1**,**C2**). However, CSA does not address potentially unreliable connections, the dynamic evolution of the fog, and provide only an SQL-like interface (**C3**,**C5**,**C4**).

In NES, we build on top of this work and combine the possible compute and storage capacities of the fog and the cloud.

Data Processing in Sensor Networks. Sensor networks (SNs) target a particular sub-area of the IoT scenario [10, 20]. In particular, these systems focus on distributed processing in a wireless sensor network (WSN) [21]. A major goal is resilience to intermittent and changing network connectivities. To this end, sensor nodes form a network to transfer sensor values through multiple hops to a root node and perform in-network data processing. Works in this area tackle efficiency (addressing **C2**) by optimizing the computation for battery lifetimes and enable filtering and aggregation queries over sensor data [10]. Moreover, they provide support for a dynamic execution environment (addressing **C5**). However, these approaches do not support more complex and general workloads, which combine multiple queries, languages, and algebras (**C4**). In addition, they do not provide strong fault-tolerance and correctness guarantees (**C3**). In NES, we leverage concepts from sensor networks and integrate them seamlessly across the Sensor, Fog, and Cloud Layers, resulting in a unified environment.

5. CONCLUSION

In this paper, we introduced NebulaStream, a general purpose, end-to-end data management system for the IoT. We showed that current systems are not yet ready for the upcoming challenges of the IoT era. We highlighted the system design of the NebulaStream platform and its design principles. Furthermore, we revealed upcoming research challenges and outlined possible solutions. Finally, we present first results that motivate the need of a new system design for upcoming IoT applications. With NebulaStream, we provide such a platform and enable new applications in different domains for the future.

6. REFERENCES

[1] A. Alexandrov et al. Implicit parallelism through deep language embedding. *SIGMOD Rec.*, 2016.

[2] G. Alonso et al. DPI: the data processing interface for modern networks. In *CIDR*, 2019.

[3] F. Bonomi et al. Fog computing and its role in the internet of things. In *MCC*, 2012.

[4] M. Hirzel et al. A catalog of stream processing optimizations. In *ACM Comput. Surv.*, 2014.

[5] K. Hong et al. Mobile fog: A programming model for large-scale applications on the internet of things. In *SIGCOMM*, 2013.

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

[7] J. Karimov et al. Astream: Ad-hoc shared stream processing. In *SIGMOD*, 2019.

[8] L. Kroll et al. Arc: an ir for batch and stream programming. In *SIGPLAN*, 2019.

[9] A. Kunft et al. Optimizing end-to-end machine learning pipelines. In *VLDB*, 2019.

[10] S. R. Madden et al. Tinydb: An acquisitional query processing system for sensor networks. In *TODS*, 2005.

[11] D. Miorandi et al. Internet of things: Vision, applications and research challenges. In *Ad Hoc Networks*, 2012.

[12] T. Neumann. Efficiently compiling efficient query plans for modern hardware. In *VLDB*, 2011.

[13] D. O'Keeffe et al. Frontier: Resilient edge processing for the internet of things. In *VLDB*.

[14] S. Palkar et al. Weld: A common runtime for high performance data analytics. In *CIDR*, 2017.

[15] G. Reinsel et al. Data age 2025: The digitization of the world from edge to core. *Journal*, 2018.

[16] M. Ryden et al. Nebula: Distributed edge cloud for data intensive computing. In *ICDE*, 2014.

[17] Z. Shen et al. Csa: Streaming engine for internet of things. In *IEEE Data Eng. Bull.*, 2015.

[18] J. Traub et al. Optimized on-demand data streaming from sensor nodes. In *SoCC*, 2017.

[19] J. Traub et al. Efficient window aggregation with general stream slicing. In *EDBT*, 2019.

[20] Y. Yao et al. The cougar approach to innetwork query processing in sensor networks. In *SIGMOD*, 2002.

[21] J. Yick et al. Wireless sensor network survey. In *Comp. Netw.*, 2008.

[22] S. Zeuch et al. Analyzing efficient stream processing on modern hardware. In *VLDB*, 2019.