Scaling a Public Transport Monitoring System to Internet of Things Infrastructures

Haralampos Gavriilidis[1] Adrian Michalke[2] Laura Mons[2] Steffen Zeuch[1,2] Volker Markl[1,2]
[1]Technische Universität Berlin [2]DFKI GmbH
{gavriilidis,volker.markl}@tu-berlin.de,{adrian.michalke,laura.mons,steffen.zeuch}@dfki.de

ABSTRACT

Applications for the Internet of Things (IoT) face several challenges when it comes to exploiting the underlying infrastructure for data management operations efficiently. IoT infrastructures consist of heterogeneous compute nodes and geographically distributed network topologies. Today's IoT applications offload data management to cloud-based stream processing engines (SPEs). However, this offloading represents a severe bottleneck that might hinder upcoming large-scale IoT applications in the future. In our demonstration, we showcase this problem using a public transport application as a potential large-scale IoT application. Our application consists of an interactive map for monitoring public transport vehicles and current demand. We implement this application on top of NebulaStream (NES), a new data management system that is designed for the IoT. In contrast to common cloud-based SPEs, NES answers queries by unifying cloud, fog, and sensor nodes under one system. Thus, NES minimizes network traffic and avoids resource over-utilization by considering the physical network topology and available compute nodes. The goal of this demonstration is to reveal the shortcomings of current system designs for large-scale IoT applications. Furthermore, we showcase how NES addresses these shortcomings and thus enables future large-scale IoT applications.

1 INTRODUCTION

Applications for the IoT, such as reporting and monitoring dashboards, consist of real-time data preprocessing and data mining tasks. Such applications visualize high-velocity data streams, resulting from large sensor networks, which flow through heterogeneous hardware and network topologies to the cloud.

Sensor streams naturally match the stream processing programming abstractions provided by cloud-based SPEs. Thus today's IoT applications use such systems to offload data management tasks. SPEs exploit the on-demand scalability of cloud resources to process compute-intensive data management workloads efficiently. However, state-of-the-art SPEs, such as Apache Flink [4], were designed for cloud environments composed of homogeneous high-performance hardware, where nodes are interconnected through high-speed network connections.

In contrast, IoT infrastructures have different characteristics regarding compute nodes and network connections [3]. In this new type of infrastructure, nodes are heterogeneous, geographically distributed, and sparsely interconnected through unstable networks. Geographically distributed sensor nodes continuously generate data, resulting in a large number of data streams with small-sized records. Intermediate nodes, also called fog nodes, provide network resources to route sensor data to the cloud. Certain nodes provide compute resources that support executing data management tasks. In particular, intermediate nodes range from low-end devices, such as system-on-a-chip devices, to high-end nodes, e.g., desktop computers and server racks. Cloud-based SPEs ignore intermediate nodes when distributing their load, hindering IoT applications from scaling beyond the cloud.

To scale data management tasks on all participating devices of an IoT infrastructure, the design of data management systems must be revisited. Recent work points out that to exploit resources of every node in an IoT infrastructure, data management systems must employ infrastructure-aware execution strategies [11]. A data management system for the IoT should leverage the scale-out capabilities of the cloud, and at the same time, exploit the resources of intermediate nodes. In particular, cloud nodes can scale resources for compute-intensive tasks within the cloud, while intermediate nodes apply fog computing techniques [6–8] to reduce intermediate results. As a result, network traffic and cloud resources are minimized. However, cloud-based approaches utilize intermediate nodes only to forward data.

NebulaStream [11] is an application and data management platform designed for the upcoming IoT era. NES addresses the mentioned IoT challenges by unifying sensor, fog, and cloud nodes into a single system. To this end, NES combines research from sensor network, distributed, and database system communities. This allows NES to transparently optimize and efficiently execute data management workloads across IoT infrastructures. The unified sensor-fog-cloud approach enables scaling the number of sensors in data management and visualization applications.

In our demonstration, we simulate an IoT infrastructure with Raspberry PIs and showcase a visualization application for a public transport system. Our application aims to provide real-time monitoring for public transport systems and suggestions for vehicle rescheduling. Therefore, it detects critical geographical areas, i.e., underserved areas with high demand. Our GUI consists of an interactive map, which visualizes real-time public transport vehicles and potential passengers. During the demonstration, visitors will interact with the map by filtering objects and configuring the clustering algorithm employed for critical area detection.

Furthermore, visitors will explore potential execution strategies for data management tasks on IoT infrastructures. We demonstrate how all participating nodes of a public transport system can be part of a query and how this influences performance and resource utilization. Our application demonstrates both centralized system designs and new designs enabled by NES. Overall, we showcase that in contrast to cloud-based SPEs, NES allows large-scale applications on IoT infrastructures, and thus enables a variety of upcoming IoT application use cases.

The rest of this paper is structured as follows. In Section 2, we discuss IoT application scenarios and challenges related to data management. In Section 3, we provide a brief overview of NES's design and architecture. After that, in Section 4, we present our demonstration scenario and finally conclude in Section 5.

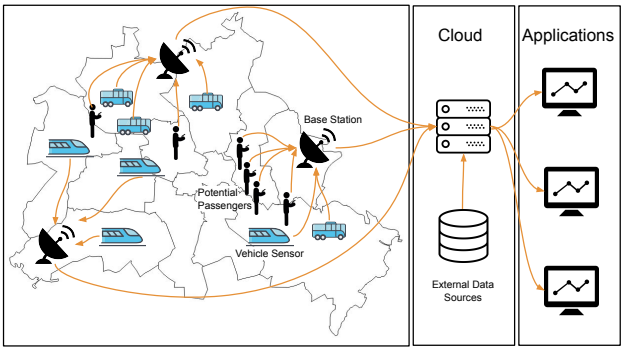

Figure 1: Exemplary IoT Infrastructure.

2 DATA MANAGEMENT IN THE IOT

In the following, we introduce a representative IoT scenario (Section 2.1) and discuss data management challenges (Section 2.2).

2.1 IoT Application Scenarios

In Figure 1, we illustrate a public transport system as a representative large-scale IoT scenario. In our scenario, the cloud hosts applications, i.e., a real-time dashboard that monitors the underlying IoT infrastructure. The infrastructure consists of geographically distributed and constantly moving sensors, base stations, and cloud nodes. In particular, potential passengers with mobile phones and vehicles with attached sensors move within a city and transmit measurements to base stations in regular time intervals. Base stations are geographically distributed nodes that collect and forward sensor streams to the cloud. Cloud nodes receive the data from the base stations and enrich it with external sources, e.g., databases with weather or air pollution data. Finally, applications consume the preprocessed sensor stream, apply additional processing, and visualize the result for users.

Public transport agencies could employ such applications to enable various smart city optimizations. Examples include rescheduling vehicles based on the demand of potential passengers, air pollution reduction by traffic light control mechanisms, and ad-hoc route planning to cope with traffic jams.

2.2 IoT Infrastructure Challenges

Today's IoT applications use data acquisition systems for data ingestion [7, 9, 10] and cloud-based SPEs for data management operations. IoT infrastructures differ significantly from cloud infrastructures, and thus pose several challenges for data management operations. Zeuch et al. [11] point out that cloud-based SPEs rely on assumptions that IoT infrastructures violate. First, fog and cloud paradigms assume different network topologies. In particular, processing nodes in cloud-based SPEs are densely connected, i.e., each node has stable connections with all other nodes. In contrast, the physical IoT topology predefines the network paths from data sources (sensors) to data sinks (cloud). Therefore, every node accesses only the subset of data that is routed through it. For example, in the IoT infrastructure depicted in Figure 1, base stations located in the west of the city are not able to directly access sensor streams that are generated on the east of the city.

Second, both paradigms expect different types of input streams. In an IoT infrastructure, millions of sensors constantly produce data streams. Those streams consist of small records that possibly capture physical phenomena, such as earthquakes, and might produce data at infrequent intervals. In contrast, cloud-based SPEs expect a few large-volume data streams at constant producing intervals. Furthermore, to ingest the sensor streams, cloud-based systems utilize message brokers, such as Apache Kafka.

In sum, IoT applications relying on cloud-based data management paradigms do not exploit intermediate nodes. The assumptions of cloud-based SPEs regarding physical topologies and types of incoming streams do not hold in IoT infrastructures.

3 NEBULASTREAM PLATFORM OVERVIEW

In the following, we present specific aspects of NES. We refer the reader to our previous work [11] for a detailed description of NES. First, in Section 3.1, we outline the limitations of cloud-based SPEs that prevent applications from exploiting IoT infrastructures. After that, we describe NES and its architecture in Section 3.2.

3.1 Limitations of State-of-the-art SPEs

In the following, we discuss two important features that hinder cloud-based SPEs to efficiently support future IoT scenarios.

Exploiting Intermediate Nodes: Applications relying on cloud-based SPEs do not exploit all participating heterogeneous intermediate and sensor nodes, since data management tasks are executed only in the cloud layer. For example, consider a simple aggregation task, e.g., counting the number of vehicles per geographical area. To execute this task, cloud-based systems would have to wait until intermediate nodes propagate data to the cloud. However, in the described IoT infrastructure, intermediate nodes can execute this task at an earlier stage and forward pre-aggregated intermediate results.

Minimizing Network Resources: If a set of running queries requires only a subset of the sensor data, acquisitional data processing avoids redundant sensor reads [7]. Another technique to further reduce network traffic between sensors and cloud is to adapt sensor sampling frequencies based on the query requirements [9]. For example, a vehicle could acquire and send its data only if it is located in a certain area [11]. These techniques are not available in cloud-based SPEs, but allow for scaling IoT data processing by minimizing redundant data traffic.

3.2 Architecture

In the following, we describe NES's architecture, illustrated in Figure 2. We focus only on the components that are related to our application scenario and omit components that are out of the scope of this demonstration, e.g., fault-tolerance mechanisms.

Optimization: NES exposes APIs for common data processing operations, as found in cloud-bases SPEs, extending those with fog-specific abstractions. IoT applications, e.g., our visualization application, submit queries to the underlying data streams, both ad-hoc and long-running. To allow for multi-query optimization, the *Query Manager* maintains a catalog of submitted queries. Query optimization and execution in NES proceeds as follows. First, NES translates user queries to logical query plans. After that, the plan is handed over to the optimizer. The *NES Optimizer* consults the *NES Topology Manager*, which provides information about the infrastructure status and the performance statistics. After optimizing the execution plan, it is handed over to the *NES Deployment Manager*.

Deployment and Execution: NES's execution plan maps segmented sub-plans to participating intermediate, or cloud nodes. Each segment contains processing instructions (tasks), as well as information about I/O operations. The NES Deployment Manager is responsible for transmitting the sub-plans to each node. After

receiving a sub-plan, a node initializes the necessary network connections and starts the execution utilizing its task scheduler.

Monitoring: During runtime, the NES Topology Manager monitors the execution, and reacts to changes incrementally, e.g., by transitioning smoothly between execution plans. To this end, the NES Topology Manager collects hardware statistics, such as CPU and main memory utilization, and additional metadata, such as selectivities and data distributions. NES nodes are designed to handle several scenarios autonomously, e.g., transient network failures. Once failed network connections are restored, topology updates are propagated to the Topology Manager.

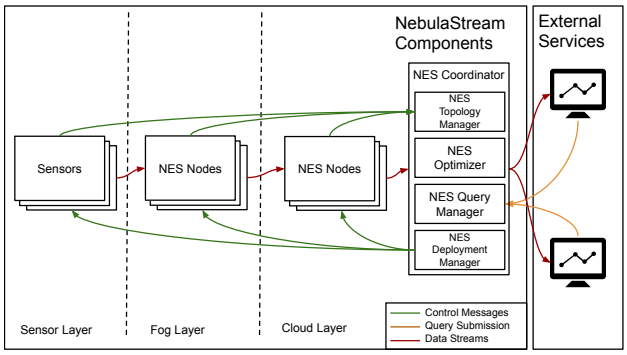

Figure 2: Simplified NES architecture overview.

The holistic view of the underlying infrastructure and the submitted queries allow NES to optimize for specific combinations of queries and topologies. In contrast, cloud-based SPEs handle incoming queries independently and schedule queries with external cluster resource management frameworks. Its characteristics allow NES to overcome limitations of cloud-based SPEs in IoT infrastructures, and to offer scalable data management for the upcoming IoT application scenarios.

4 DEMONSTRATION

In the following, we describe our user interface (Section 4.1), demonstration setup (Section 4.2), and application (Section 4.3).

4.1 User Interface

Visitors interact with our application through a GUI, as shown in Figure 3. Our GUI consists of an interactive map, which includes moving objects that are either potential passengers or public transport vehicles (trains, buses, etc.). Our application aims at detecting crowded geographical areas, that public transport vehicles underserve. The application clusters potential passengers according to their geolocation. When public transport vehicles underserve a cluster of potential passengers, our application notifies the visitor by highlighting the respective clusters on the map. The resulting notifications are useful for public transport agencies, e.g., to schedule vehicles for crowded areas.

Additionally, our GUI allows filtering by moving the visible map area and by selecting vehicle types. The visitor configures the visible objects and the clustering algorithm by changing parameters, e.g., object distance and cluster size, as shown in Figure 3 (1). A main feature is choosing between processing modes (2) that represent the solution space for IoT applications. To show the implications of each mode, we provide performance metrics, such as ad-hoc application statistics (3) and resource utilization (4). Our demonstration aims at showcasing the strengths and weaknesses of each solution in a hands-on experience.

4.2 Demo Setup

In the following, we describe the setup of our demonstration.

Hardware: For our demonstration scenario, we assume a topology where each public transport vehicle carries a sensor transmitting information to base stations at regular time intervals. Potential passengers send their geolocation to base stations, e.g., through a mobile application. We use Raspberry PIs as base stations (fog nodes), and a mobile workstation as a cloud node.

Software: Our application consists of a frontend and a backend component. The backend uses NES to offload data management operations. The backend acts as a sink for NES, i.e., it is an intermediator between multiple user frontends and NES. It coordinates query transmission to NES, forwarding results to connected frontends. Our application backend is implemented in Python and utilizes the Flask microframework. The frontend is implemented in Javascript, utilizes websockets for communication, Leaflet for its map, and Grafana for monitoring.

Dataset: We simulated two sensor streams to compose our dataset. First, we simulated vehicle sensor data using real-world *General Transit Feed Specification* datasets [1] (enhanced with vehicle occupancy). Second, we simulated potential passengers using the *Simulation of Urban Mobility* (SUMO) generator [2]. We merge the two resulting datasets, and partition them by geolocation, to resemble geographically distributed base stations. We utilize sensor data from the city of Berlin, however, our application supports all sensor streams following the GTFS schema.

4.3 Application

We highlight two aspects of an IoT data management application scenario. First, we describe our deployment process, during which our application transforms user queries to execution plans and deploys them on the underlying nodes. Second, we demonstrate several execution strategies by deploying different execution plans and revealing their implications on resource utilization.

4.3.1 Query and Deployment. Every time a visitor interacts with the map and its options, our application sends NES a new query with the following parametrized operations.

Map Bounding Box: These parameters filter vehicles and potential passengers located within a bounding box. The bounding box is defined by two coordinates and is automatically computed by the map area currently viewed by the visitor.

Vehicles and Passengers: These parameters filter potential passengers and selected vehicle types, e.g., bus, train, or subway.

Clustering: These parameters, e.g., object distance and cluster size, are related to our clustering algorithm (DBSCAN [5]).

Overall, our application composes and deploys queries that filter sensor records, cluster them by geolocation, calculate the average vehicle capacity within each cluster, and yield critical areas depending on user-defined thresholds.

In this application, NES allows us to reduce data as early as possible in the IoT infrastructure. Especially in visualization scenarios, data reduction is a key performance factor, and naturally occurs because users are seldomly interested in all sensor data. To this end, NES provides the option to filter data needed for visualization purposes already in the intermediate (fog) layer. The resulting data reduction is two-fold. First, NES uses on-demand data acquisition techniques to only gather sensor data that is currently required to answer a query. Second, NES uses intermediate nodes to evict unnecessary data close to the sensors. Both data reductions minimize the overall network traffic within the IoT infrastructure, as well as data that has to be sent to applications.

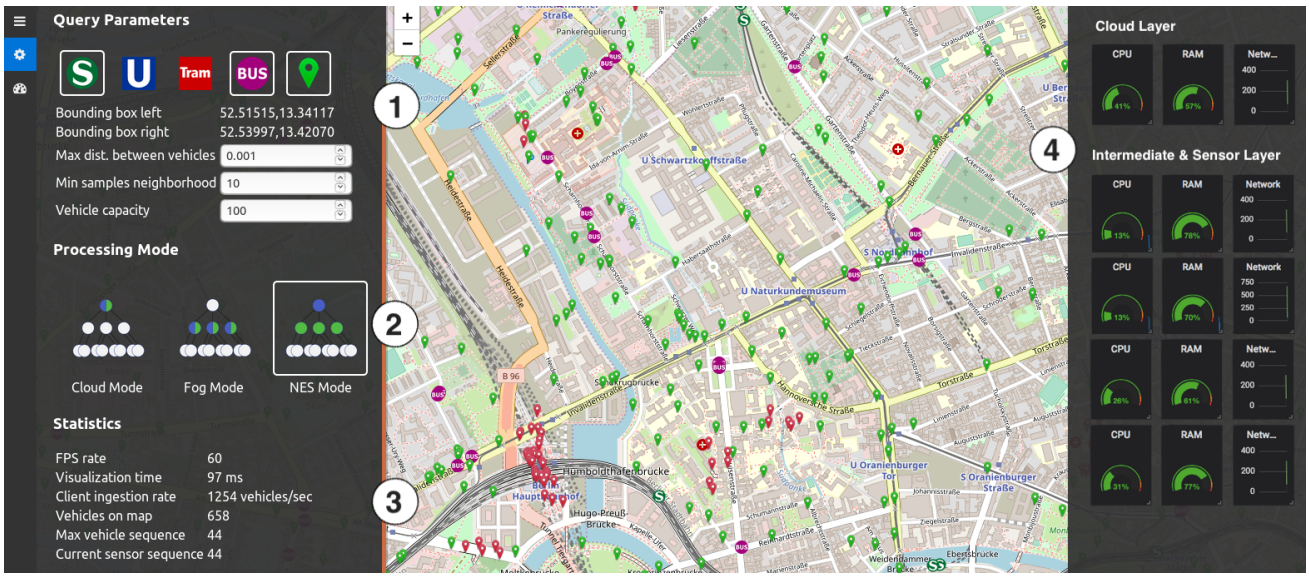

Figure 3: Demo GUI with potential passengers, buses, and trains. The application clusters potential passengers (green) by area density, and marks clusters red for insufficiently covered areas. The visitor may configure the query parameters ①, the processing modes ②, and observe query information ③ and runtime statistics about resource utilization ④.

4.3.2 Query Execution. In Section 3, we introduced the NES Optimizer, which produces, and evaluates potential execution plans. Each execution plan contains a mapping between NES operators and nodes in the IoT infrastructure. The optimizer would come up with three execution plans that resemble cloud-based, fog-based, and unified approaches, which we refer to as processing modes. Note that in our demonstration, we use NES to reproduce all processing modes.

The graphs in Figure 3 ② illustrate the processing modes. Vertices represent sensor, fog and cloud nodes, while edges define the network connections between them. We color filter operations green, and clustering operations blue. In our demonstration, visitors can choose between the following three processing modes.

Cloud Mode: NES places all operations on the cloud, and utilizes the remaining nodes only for data forwarding. Performance metrics will reveal that the main workload gathers in the cloud layer, while resources of intermediate fog nodes remain unused.

Fog Mode: NES places all operations on the intermediate layer. Filtering on the fog reduces network traffic. CPU, and RAM utilization on the fog nodes increases significantly.

NES Mode: NES places filter operations on fog nodes and clustering operations on the cloud. Performance metrics show that network traffic, CPU and RAM utilization remain at moderate levels, since operations are evenly distributed.

In sum, our demonstration showcases how future data management systems allow exploiting all resources of an IoT infrastructure, instead of relying solely on the cloud. Our public transport monitoring system and its underlying topology resemble common IoT application scenarios.

5 CONCLUSION

In this paper, we highlighted data processing challenges in the IoT domain, and described a representative IoT scenario, a public transport system. Our application, a public transport monitoring system, visualizes vehicles and potential passengers on an interactive map, and detects underserved areas. Our application translates user actions on its GUI to NES queries. We offload data management tasks on the IoT infrastructure in different ways, using potential NES execution plans. The visitor can explore these execution plans and observe their implications on resource utilization. Our demonstration shows that existing systems do not address scaling data management on IoT infrastructures.

ACKNOWLEDGMENTS

This work was funded by the German Ministry for Education and Research as BIFOLD - Berlin Institute for the Foundations of Learning and Data (ref. 01IS18025A and ref 01IS18037A). The authors would like to thank the NebulaStream team for their insightful comments and fruitful discussions.

REFERENCES

[1] 2019. General Transit Feed Specification. https://gtfs.org/. Accessed: 2019-11-22.
[2] 2019. SUMO - Simulation of Urban Mobility. http://sumo.dlr.de/index.html. Accessed: 2019-11-14.
[3] Flavio Bonomi et al. 2012. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 13–16.
[4] Paris Carbone et al. 2015. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 36, 4 (2015).
[5] Martin Ester et al. 1996. A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96)*. 226–231.
[6] Alberto Lerner et al. 2019. The Case for Network Accelerated Query Processing. In *CIDR*.
[7] Samuel R Madden et al. 2005. TinyDB: an acquisitional query processing system for sensor networks. *ACM Transactions on database systems (TODS)* 30, 1 (2005), 122–173.
[8] Dan O'Keeffe et al. 2018. Frontier: Resilient edge processing for the internet of things. *Proceedings of the VLDB Endowment* 11, 10 (2018), 1178–1191.
[9] Jonas Traub et al. 2017. Optimized on-demand data streaming from sensor nodes. In *Proceedings of the 2017 Symposium on Cloud Computing*. ACM, 586–597.
[10] Jonas Traub et al. 2019. SENSE: Scalable Data Acquisition from Distributed Sensors with Guaranteed Time Coherence. *arXiv preprint arXiv:1912.04648* (2019).
[11] Steffen Zeuch et al. 2020. The NebulaStream Platform: Data and Application Management for the Internet of Things. In *CIDR*.