Data Management for the Internet of Things: Building a Public Transport System Application

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

Current cloud-based data management systems are not designed for the needs of the upcoming Internet of Things (IoT) applications. In particular, they do not consider heterogeneous setups regarding network topologies and compute nodes. In this demonstration, we showcase one representative use case for the IoT, i.e., a public transport system of a city, and describe its unique challenges. We discuss why state-of-the-art data management systems do not qualify for this task and demonstrate the NebulaStream (NES) as a system designed for IoT applications. NES unifies data processing techniques from cloud and fog-based systems. By considering the physical network topology and available compute resources, NES minimizes network traffic, and avoids resource over-utilization. Our application on top of NES allows visitors to deploy ad-hoc queries and explore different processing modes. Overall, we show that NES enables future IoT applications that are not realizable with current state-of-the-art systems.

1 INTRODUCTION

Data analytics applications for the Internet of Things (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.

Today's IoT applications use cloud-based stream processing engines (SPEs) for their data management tasks, since sensor data matches naturally the provided stream processing abstractions. Furthermore, SPEs exploit the on-demand scalability of cloud-resources to support compute-intensive data management workloads. 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, both regarding the nodes and the network connections [3]. Sensor nodes, potentially geographically distributed, continuously generate data, which result in a large number of data streams with small-sized records. Intermediate nodes route the produced sensor data to the cloud. In this new type of infrastructure, intermediate nodes are heterogeneous, geographically distributed, and sparsely interconnected through unstable networks. In particular, IoT devices range from low-end nodes, such as system-on-a-chip devices and cheap sensors, to high-end nodes, such as desktop computers and server racks.

Hence, using cloud-based SPEs for IoT applications restricts scaling data management operations only within 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 infrastucture-aware execution strategies [9].

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. The set of intermediate nodes, also known as fog [3], are only used for data forwarding when using cloud-based systems for data management. In particular, the cloud can scale resources for compute-intensive tasks, while nodes in the fog can apply in-network processing [6], fog computing techniques [8] and acquisitional data processing [7], to reduce intermediate results. By reducing intermediate results, network traffic and cloud resources are minimized.

NebulaStream (NES) [9] 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. NES combines research from sensor networks, distributed and database systems communities. This allows NES to transparently optimize and efficiently execute data management workloads across IoT infrastructures.

Overall, this unified cloud-fog approach enables to scale the number of sensors in data management and visualization applications, while efficiently exploiting the existing hardware infrastructure. We demonstrate a visualization application for a public transport system on top of NES. Our application aims to provide real-time monitoring for public transport systems, and detects geographical areas that are critical, i.e. not sufficiently covered by public transport. Its GUI consists of an interactive map, which visualizes real-time public transport vehicles and potential passengers. Through the GUI, visitors are able to move the map, filter vehicles, and configure the parameters for the critical area detection. To detect underserved areas, we use a density-based clustering algorithm. Public transport systems could use this information to reschedule vehicles manually, or even automatically.

Using the public transport of a city as a representative IoT application, we demonstrate how all participating nodes can be part of a query, and how this influences performance and resource utilization. Overall, we showcase that 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 challenges related to data management in IoT infrastructures. In Section 3, we provide a brief overview of NES's design principles and its architecture. After that, in Section 4, we present our demonstration scenario and finally conclude in Section 5.

2 DATA MANAGEMENT IN THE IOT

Today's data management application scenarios in the IoT cover a broad range, e.g., failure prediction and anomaly detection in factories, training models for self-driving cars, or tracking of supply chains. In the following, we present a representative IoT scenario and describe the challenges associated with query

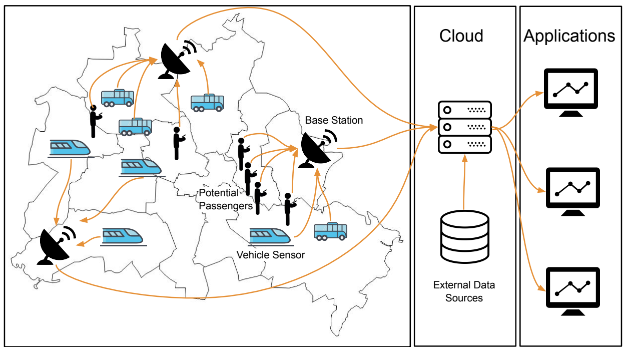


Figure 1: Common IoT Infrastructure.

processing. In general, IoT scenarios comprise moving or static sensors connected to the cloud through multiple intermediate nodes.

2.1 IoT Application Scenarios

In Figure 1, we show a public transport system as a representative IoT application. In this system, vehicles and potential public transport passengers with attached sensors move around the city. Sensors transmit their data in regular time intervals to geographically distributed base stations, which forward sensor data to the cloud. The base stations, as well as the vehicles, represent intermediate nodes in the IoT infrastructure. On the cloud, sensor streams are subject to enrichment from external sources, e.g., weather or air pollution data. The resulting data streams are then fed into applications, to answer ad-hoc user queries, to visualize data in GUIs, or to perform advanced analytics. If a city would implement such an IoT infrastructure, it would enable smart city optimizations such as:

- Reduce air pollution by traffic light control mechanisms.
- Ad-hoc route planning to cope with sudden traffic jams.
- Automate schedule updates for the public transport based on crowdedness.

2.2 IoT Infrastructure Challenges

In the following, we discuss challenges related to data management in the IoT. Zeuch et al. [9] point out, that cloud-based SPEs rely on assumptions which are violated in IoT infrastructures. First, both paradigms introduce different network topologies. In particular, processing nodes in cloud-based SPEs are densely connected, i.e., each node can communicate with all other nodes. In contrast, IoT infrastructures follow a structure similar to p2p networks, where the physical network topology predefines the paths from data sources (sensors) to data sinks (cloud). Therefore, every node accesses only the subset of data routed through it. For example, the base station located in the west of the city is not able to directly access sensor streams that are generated on the east side of the city. In today's IoT infrastructures, there are orders of magnitude more sensors than nodes. Hence, this physical setup forms a tree-like graph where data flows from sensors among the intermediate nodes to a sink in the cloud. Finally, when considering multiple layers of intermediate nodes, there might be multiple paths from the source to the cloud. Second, both paradigms expect differently sized input streams. In the fog infrastructure, millions of sensors constantly produce small data streams that capture physical phenomena, such as earthquakes. In contrast, cloud-based SPEs are built around the assumptions that few, large data streams or utilize data broker (e.g., Kafka) to produce them as an input.

3 NEBULASTREAM PLATFORM OVERVIEW

In this demonstration, we focus on specific aspects of NES and provide a brief overview. For a detailed description of NES, we refer the reader to our previous work [9]. First, we outline the limitation of state-of-the-art cloud-based SPEs that prevent them from exploiting upcoming IoT infrastructures Section 3.1. After that, we describe NES and its architecture in Section 3.2.

3.1 Limitation of State-of-the-art SPEs

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

Exploitation of Intermediate Nodes: Intermediate nodes route the sensor-generated data to the cloud. In an IoT infrastructure, devices are heterogeneous. They range from low-budget processing devices, such as mobile phones and Raspberry PIs, to standard desktop computers and high-end processing nodes with GPUs. Current SPEs are unable to exploit all intermediate nodes since applications must wait until generated data reaches the cloud. For example, consider a simple aggregation task, such as counting the number of potential passengers per geographical area. To execute this task, cloud-based systems would have to wait until intermediate nodes propagate the data to the cloud. However, in the described IoT infrastructure, intermediate nodes can execute this task before the data reaches the cloud.

On-demand Data Acquisition: If the set of running queries does not require all incoming sensor data streams, it would be possible to avoid these sensor reads by employing acquisitional data processing [7]. Additionally, by adapting the sampling frequency of each sensor based on the query requirements, we can reduce network traffic between sensors and the cloud further. For example, a vehicle could potentially acquire and send its data only if it is located in a certain area.

3.2 Architecture

In the following, we describe NES's architecture. We illustrate the architecture in Figure 2, and focus only on the components that are related to our application scenario. Note that NES provides consists of additional components, but we omit their descriptions because they are out of the scope of this demonstration. We refer the reader to our recent work [9] for more details.

Optimization Process: External Services, e.g., the visualization application from our scenario, submit queries to NES. NES provides APIs to describe common data processing operations, similarly to state-of-the-art SPEs but with fog-specific extensions. The *Query Manager* manages and coordinates incoming queries. Query execution in NES proceeds as follows. First the user queries are translated to logical query plans. After that, the plan is handed over to NES's Optimizer. The *NES Optimizer* consults the *NES Topology Manager*, which monitors the infrastructure changes and performance statistics. The NES Optimizer then composes the execution plan, which is then deploys to its nodes through the *NES Deployment Manager*. In case of new queries or topology updates, the NES Deployment Manager applies required changes in an incremental fashion.

Deployment and Execution: NES's execution plan maps segmented sub-plans to participating processing intermediate or cloud nodes. Each segment contains processing instructions (tasks) as well as input and output information. The NES Deployment Manager is responsible for transmitting the sub-plans to each node. After receiving a sub-plan, a node sets up the necessary connections with other nodes and starts the execution. Each node has its own task scheduler, which uses a thread pool and assigns tasks alongside I/O operations to its resources.

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 or main memory usage and application-specific statistics like selectivity or data distributions. In NES, nodes are designed to handle a wide range of scenarios autonomously. For example, in case of transient network failures, a node is able to change buffer execution strategies or buffer processed data. Once the failed network connections are restored, the changes are propagated to the NES Topology Manager.

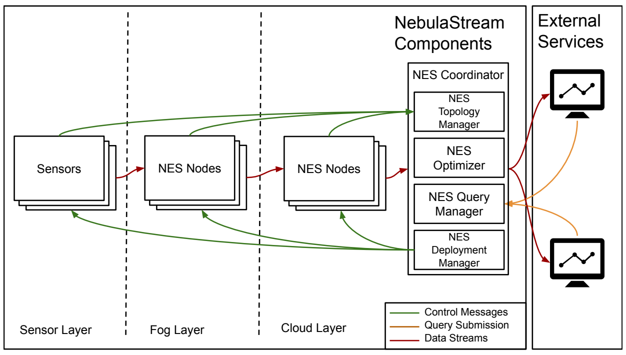


Figure 2: Simplified NES architecture overview.

Overall, its characteristics allow NES to overcome the limitations of cloud-based SPEs. NES paves the way for a scalable data management back-end for applications in the IoT domain.

4 DEMONSTRATION

We demonstrate NES through an application for public transport. In the following, we describe our user-interface (Section 4.1), the setup (Section 4.2), and the application (Section 4.3).

4.1 User Interface

Visitors interact with our application through a GUI, as shown in Figure 3. The 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 areas that are covered insufficiently by public transport service and thus should be re-scheduled by the public transport agency. To this end, the application clusters potential passengers according to their geolocation. When public transport vehicles do not sufficiently cover a cluster of potential passengers, our application notifies the visitor by highlighting map markers.

Besides its notification mechanism, our GUI allows filtering objects by moving the visible map area and by selecting vehicle types. The visitor may configure the filters and the clustering algorithm by changing parameters Figure 3 ①, such as object distance and cluster size. Furthermore, our GUI also includes performance metrics, such as ad-hoc application statistics ③ and resource utilization ④. A main feature of our GUI is choosing between processing modes ②. The modes represent the solution space for IoT applications. We aim to showcase the strengths and weaknesses of each solution in a hands-on experience.

4.2 Demo Setup

In the following describe the setup of our demonstration.

Hardware: For our demonstration scenario, we assume a topology where each public transport vehicle carries a sensor, which transmits its geolocation to a base station, at regular time intervals. Potential passengers also send their geolocation to the base stations, e.g. by a mobile application that uses the sensor attached to a mobile phone. For demonstration purposes, we use Raspberry Pis as base stations (fog nodes) and a Laptop as a cloud node. The nodes are interconnected through a router.

Software: Our application consists of a user frontend, and a webserver which uses NES as its execution engine for data processing. The webserver acts as an intermediator between multiple user frontends and NES. It coordinates query transmission to NES and forwards query results to our frontends. Our application backend is written in Python, and uses the Flask microframework. The frontend is implemented in Javascript and uses websockets to communicate with the webserver. We use the Leaflet framework to implement the interactive map.

Dataset: We simulated two sensor types to compose the dataset for our demonstration. First, we simulate vehicle sensor data directly at the base stations using real-world *General Transit Feed Specification* datasets [1]. Second, we simulate potential passengers using the *Simulation of Urban Mobility* (SUMO) generator [2]. We partition both datasets by geolocation, to resemble geographically distributed base stations. The sensor records contain the measured time and information about each measurement, such as sensor geolocation and vehicle type.

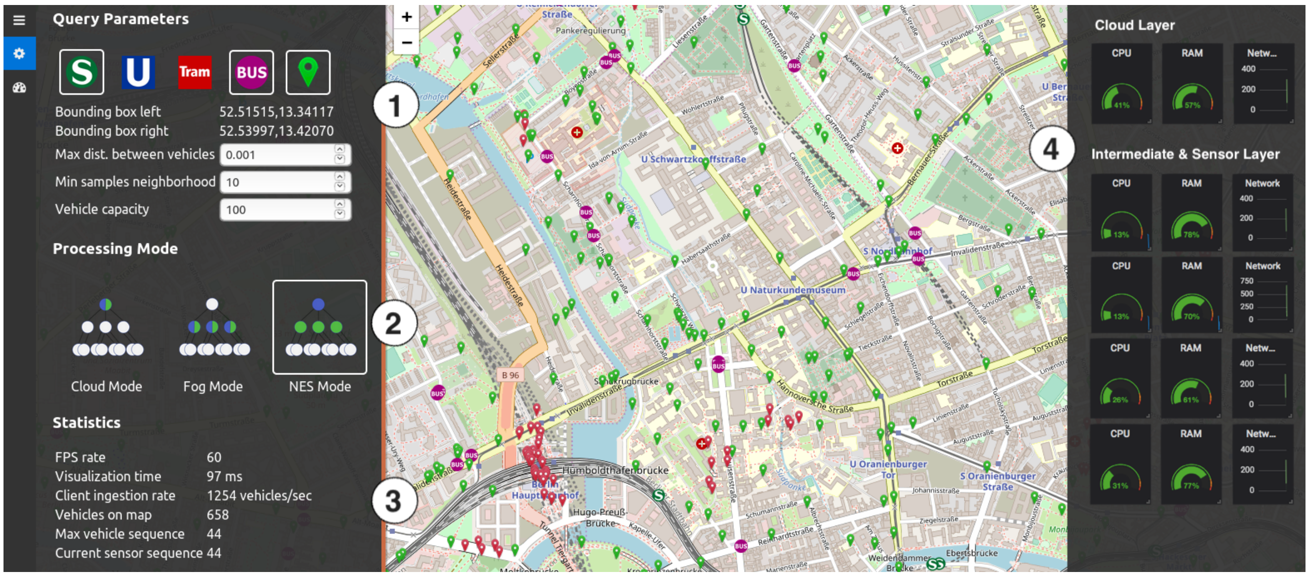
4.3 Application

In our demonstration, we highlight two aspects of a data management system for the IoT. First, we showcase the deployment process that takes submitted ad-hoc queries from a user interface as input and deploys operators on the processing nodes to answer this query. Secondly, we showcase the query execution process in NES, by deploying different execution plans and revealing their implications on resource utilization.

4.3.1 Query Deployment.

The main goal of our application is to detect underserved areas based on crowdedness. 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 the entire data set. 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 the query. Seconds, NES uses intermediate nodes to evict unnecessary data close to the sensors. Both data reductions minimize the overall network traffic inside the IoT infrastructure, as well as data that has to be sent to applications, in our case the web server.

In our demonstration, the application sends a new query to NES, every time a visitor interacts with the map and its options. A visitor interaction on the map creates a new query that consists of the following parametrized operations:


Query Parameters

Bounding box left	52.51515,13.34117
Bounding box right	52.53997,13.42070
Max dist. between vehicles	0.001
Min samples neighborhood	10
Vehicle capacity	100

Processing Mode

Cloud Mode Fog Mode NES Mode

Statistics

FPS rate	60
Visualization time	97 ms
Client ingestion rate	1254 vehicles/sec
Vehicles on map	658
Max vehicle sequence	44
Current sensor sequence	44

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).

- **Map Bounding Box:** This parameter filters the vehicles and potential passengers, which are located within a bounding box. The bounding box is defined by the map area currently viewed by the visitor.
- **Vehicles and Passengers:** This parameter filters passengers and selected vehicle types, e.g., bus, train, or subway.
- **Clustering:** This parameter adjusts the parameters of our clustering algorithm (DBSCAN [5]), such as object distance. The algorithm clusters the previously filtered sensor records and yields the underserved areas.

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 of NES operators to 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. In Figure 3 (2), we present these modes and mark filter operations with green colour and clustering operations with blue colour. In our demonstration, visitors can choose between the following three processing modes:

- **Cloud Mode:** The cloud mode represents the processing of current cloud-based SPEs. NES places all operators on the cloud nodes. Performance metrics will reveal that the main workload gathers in the cloud layer while the processing resources of the intermediate fog nodes remain unused.
- **Fog Mode:** NES places all operators on the intermediate layer. The visitor will observe that filtering on the fog reduces network traffic. However, CPU and RAM usage on the fog nodes increases significantly.
- **NES Mode:** NES places the filter operators on the fog nodes and the clustering operator on the cloud nodes. The visitor will observe that network traffic, CPU and RAM utilization remain at a moderate level due to evenly distributed workloads alongside the early filters.

In sum, in this demonstration, we highlight the shortcomings of state-of-the-art systems for IoT applications and the benefits of using NES as a data management platform for IoT scenarios.

5 CONCLUSION

In this paper, we highlighted data processing challenges in the IoT domain and demonstrated NES, a system that tackles those challenges. We showcased NES through a visualization application for a public transport monitoring system. Our application translates user actions on its GUI to NES queries. Using different execution plans, the visitor observes the implications on resource utilization of NES's cross-paradigm operator placements. Using NES as our data management backend, we are able to scale applications in large IoT infrastructures.

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 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)*. AAAI Press, 226–231. http://dl.acm.org/citation.cfm?id=3001460.3001507
[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] Steffen Zeuch et al. 2020. The NebulaStream Platform: Data and Application Management for the Internet of Things. In *CIDR*. to appear.