Towards Context-Aware and Dynamic Management of Stream Processing Pipelines for Fog Computing

Patrick Wiener, Philipp Zehnder, Dominik Riemer
FZI Research Center for Information Technology
Karlsruhe, Germany
{wiener, zehnder, riemer}@fzi.de

***Abstract*—Newly arising IoT-driven use cases often require low-latency analytics to derive time-sensitive actions, where a centralized cloud approach is not applicable. An emerging computing paradigm, referred to as fog computing, shifts the focus away from the central cloud by offloading specific computational parts of analytical stream processing pipelines (SPP) towards the edge of the network, thus leveraging existing resources close to where data is generated. However, in scenarios of mobile edge nodes, the inherent context changes need to be incorporated in the underlying fog cluster management, thus accounting for the dynamics by relocating certain processing elements of these SPP. This paper presents our initial work on a conceptual architecture for context-aware and dynamic management of SPP in the fog. We provide preliminary results, showing the general feasibility of relocating processing elements according to changes in the geolocation.**

***Index Terms*—fog computing, distributed systems, context-aware computing, stream processing**

I. Introduction

The proliferation of the Internet of Things (IoT) and the corresponding progression in the installation of smart devices leads to a deluge of generated data. However, oftentimes the real value of IoT does not result from raw data itself, but from derived insights, that generally facilitate time-sensitive actions. Thereby, todays' applications are typically deployed in centralized, scalable cloud environments, that are specifically tailored for the needs of the respective use cases. In this regard, a popular approach is to model such applications as analytical stream processing pipelines (SPP) known as data flow programming [1], [2]. Thus, the overall output is obtained by performing various sequential transformations to the input. However, there are some inevitable downsides to solely deploying SPP in the cloud, particularly in scenarios that require time-sensitive decisions, are limited in available bandwidth, or where privacy is a limiting factor. To mitigate these issues, an obvious approach is to extend the cloud and move certain computation closer to where the data is generated commonly referred to as fog computing [3]. Hence, the use of container technologies, e.g., Docker[1], and existing cluster management solutions, e.g., Kubernetes[2], proof to be a natural fit in resource-constrained environments due to their small overhead and fast deployment [4]–[6]. Still, there is little adoption of the fog computing paradigm in practice [7].

[1]https://www.docker.com/ [last accessed: 18 March, 2019]
[2]https://kubernetes.io/ [last accessed: 18 March, 2019]

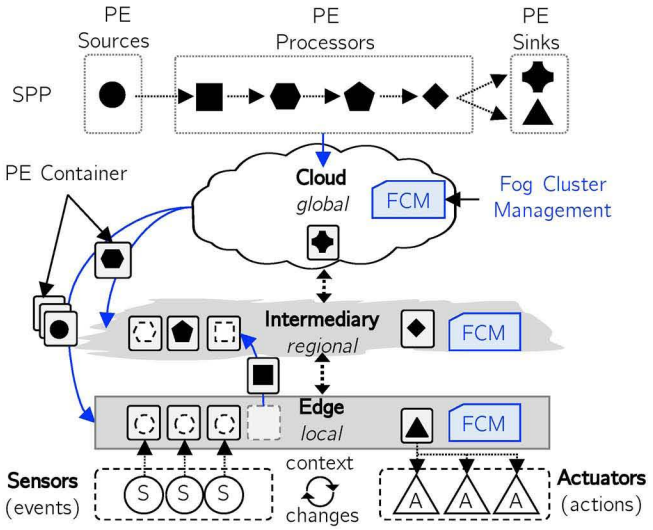


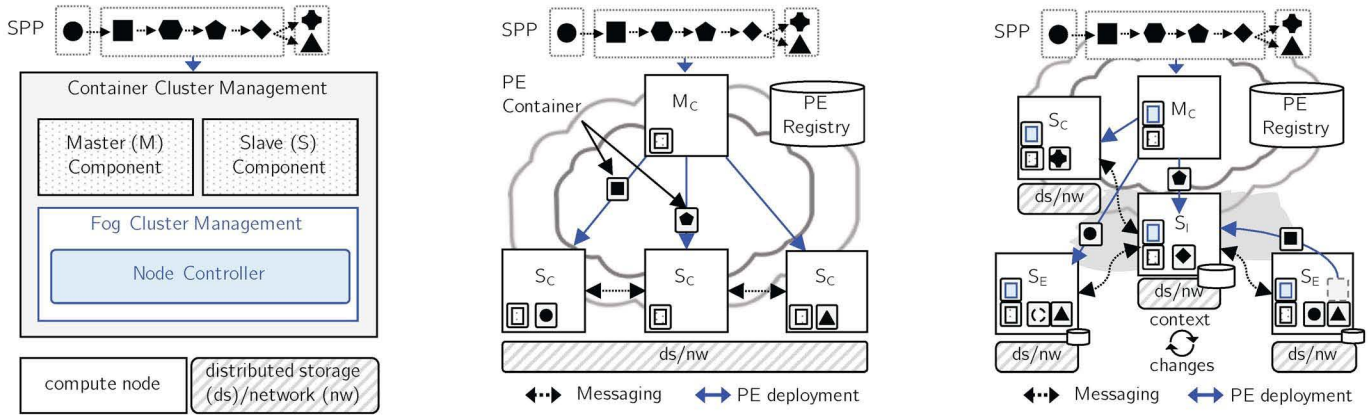
Fig. 1. Dynamic management of a stream processing pipeline (SPP) consisting of various processing elements (PE) in a fog computing infrastructure.

One reason might be the actual shortcoming of managing the multi-layered (cloud, intermediary, edge), hierarchical and heterogeneous fog computing infrastructures (FCI), as shown in Figure 4. Especially, in cases where edge nodes are constantly changing their location, the inherent context changes need to be incorporated in the underlying FCI management, thus relocating dedicated processing elements (PE) along the cloud-edge continuum when necessary. As a result, it is crucial to provide mechanisms, that deal with these situational changes in data flow and processing.

This paper presents our initial work to realize context-aware and dynamic management of FCI with respect to SPP. This involves the SPP deployment while satisfying predefined requirements, as well as the dynamic relocation of certain PE in case of context changes, e.g., location-based changes (geo-context).

The rest of the paper is organized as follows: Section II describes a motivating use case and derives requirements. Section III presents the conceptual architecture. Section IV shows preliminary results and we provide related work in Section V. Lastly, we conclude and present future work in Section VI.

978-1-7281-2365-3/19/$31.00 ©2019 IEEE



(a) high-level FCI approach (b) SPP in traditional cloud cluster (c) SPP in FCI cluster

Fig. 2. High-level FCI approach, cloud-based container orchestrators without context-awareness and context-aware SPP management in a FCI.

II. Motivating Use Case: Urban Logistics

A well-known challenge and cost factor for courier, express, and parcel service providers is the so called last mile problem, which refers to the last mile of delivery for goods from distribution hubs to end users. Thereby, the constantly growing e-commerce business further increases the traffic density in urban areas, thus, introducing various issues for cities, e.g., due to noise, emissions, or general traffic level. To mitigate these problems, we envision electrified delivery robots equipped with various sensors, e.g., laser scanners, and actuators, e.g., servo motors, that autonomously pick up packages at dedicated hubs and deliver them to the customers. By using a fog computing approach, existing intermediary and edge compute resources are exploited to orchestrate user-defined SPP to perform dedicated monitoring and analytical tasks.

A. Location Monitoring Pipeline

For the most part, delivery robots are operating autonomously and delivering packages according to a centrally planned tour. Therefore, each one is assigned to a dedicated urban district. Still, it is crucial to constantly monitor the fleet in order to immediately initiate actions in terms of any malfunctions. Consequently, it is necessary to monitor the actual location with respect to the given geofence of the dedicated urban area, thereby informing the fleet supervisor about any suspicious behaviour, that involves unintentionally leaving the district. This analytical task involves to constantly verify, whether the delivery robot is still located inside the given geofence. While generally performed locally on the robot itself, in case of violation, the algorithm should be offloaded to a nearby intermediary node.

B. Requirements

The presented use case is one example to motivate a wide field of fog applications. Thus defining specific functional and non-functional requirements that shape our systems' design approach is not feasible, which is why we refer to the more general term of requirements in the following.

SPP developers should be abstracted away from the complexity of manually managing the infrastructure by providing a holistic FCI management approach (R1). Thereby, allowing them to centrally model and deploy their SPP and specify PE requirements, that need to be satisfied, e.g., certain processing is bound to a specific region. The approach includes a seamless integration of compute resources, distributed storage and network along the cloud-edge continuum (R2). Additionally, the architecture has to account for context changes to increase the robustness, e.g., induced by mobility aspect of edge nodes, node failures, or network outages (R3). Thus, it is important to have a component running on each participating node that observes the context and provides decentralized decision-making capabilities to some extent (R4). PE are distributed using lightweight container technology due to their ease of deployment, remote execution ability, portability and compatibility (R5).

III. Conceptual Architecture for Fog Computing Infrastructures

In the following, we briefly elaborate on the application model, that lays the foundation for the conceptual architecture. We then give a high-level overview of our approach, compare it to existing cloud-based container orchestrators without context-awareness and present our system model for context-aware SPP management in FCI.

A. Application Model

Our application model for SPP is based on the dataflow programming pattern [1], [2]. As shown in Figure 3, SPP are composed of (1) processing elements (PE), that are each responsible for a specific task, e.g., data source/sink adapters, numerical filters, or advanced machine learning classifiers, and (2) their interconnections, that are based on the publish-subscribe pattern [8]. The segregation of complex analytical tasks in smaller, decoupled computational units allows for a modular, flexible, and reusable SPP design. Individual PE can be realized in a variety of different programming languages

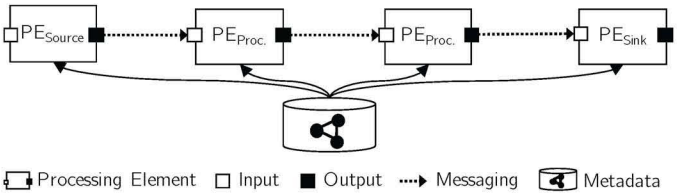


Fig. 3. Sample stream processing pipeline topology.

and frameworks (e.g. Java, Apache Flink[3], Python). Therefore, it is necessary that each PE is semantically described in order to use the metadata such as topics to publish/subscribe, input/output data types and format it expects/provides to enable PE matching. This modular application design paired with lightweight containerization provide a feasible approach for packaging and deployment of PE container including application logic, runtime environment as well as configuration on the underlying FCI [4], [5].

B. System Model

A common architectural pattern for designing container-based systems evolves around the master-slave architecture as shown in Figure 2a. Such systems have at least one master and one or more slave processes running as containers on different compute nodes. Besides various other infrastructure services, e.g., for distributed storage, or networking, these components make it possible to abstract the complexity of infrastructure management in terms of scheduling, deployment, scaling away from the actual predesign and modelling phase of SPP, thus alleviating the developers work. The master component monitors the state of various deployments and the actual infrastructure and is further responsible to keep the target and actual status in sync. Whenever a deployed PE container dies, the master intervenes and tries to bring this container back up by invoking re-deployment actions on the slave nodes. Therefore, slave nodes constantly report their state and resource capacity to the master to be taken into account for deployments or potential relocations. Typically, master and slave components interact with each other via REST-based services. In addition, we see our node controller to be a supplementary component in this ensemble of infrastructural containers in order to account for the context changes, that are characteristic for the presented use case.

In contrast, state of the art solutions for container management, such as Kubernetes, typically operate in cloud clusters as depicted in Figure 2b. Generally, the application descriptions are passed to the master component in the form of a deployment manifest, that also contains user-defined requirements in order for the scheduler to decide, where to deploy the container. The orchestrator then receives this information and starts the container on the dedicated slave nodes based on the container image, that resides is a registry. On a logical level, all compute nodes reside in one geographic cloud environment.

[3]https://flink.apache.org/ [last accessed: 18 March, 2019]

Thereby, the cluster is interconnected via dedicated software-defined networking solution as well as distributed storage in order to account for stateful applications.

In a fog computing infrastructure, compute nodes are spread along a multi-layer, hierarchical infrastructure, that is geographically distributed and heterogeneous in terms of node types and compute resources as indicated in Figure 2c. Nevertheless, these compute nodes are also classified with respect to the master-slave terminology, but are additionally differentiated more finely according to their specific geolocation. While there still exists some sort of interconnected software-defined, overlay network and distributed storage solution, a FCI management solution must account for managing the inherent dynamics induced by the mobility aspect of edge nodes, node failures, network outages, etc. This is where we envision our node controller component to fit in, in order to soften the strong centralized management approach of existing solutions towards a decentralized and context-aware management when needed. We introduced our vision of a holistic fog cluster

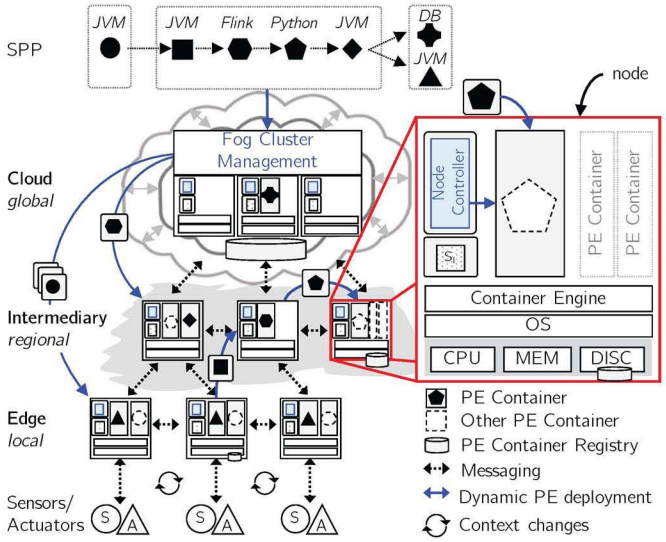


Fig. 4. Context-aware and dynamic management of stream processing pipelines in a fog computing infrastructure [9].

management [9], that spans along the cloud-edge continuum as shown in Figure 4. Typically, the cluster management is implemented leveraging container technologies, where there exists an intermediary layer (often referred to as fog), that provides additional processing power in closer proximity to the edge as compared to the cloud [7]. At the edge, nodes typically have access to sensors and actuators to retrieve data or trigger actions with respect to the real-world such as the delivery robot performing evasive maneuvers according to measurements from optical sensors. The cloud functions as a centralized scalable backbone for all sublayers, e.g., to store analyzed and aggregated data, to run big data analytics or to provide adequate compute resources for certain PE of the SPP. Furthermore, each PE defines infrastructural requirements, e.g., hardware requirements (cpu, memory, storage, physical

interfaces of a node) and software requirements (compatibility with container runtime version, message broker version) as well as various other extensible user-defined requirements, e.g., privacy-zones. The node controller runs on every node in the FCI and is responsible (1) to watch for context changes, e.g., system-context due to shared resources on compute node, location-context due to geographical restrictions, temporal-context due to time restrictions, (2) to re-deploy a PE, when the central cluster manager is not available due to connectivity issues, and (3) to relocate specific PE container. To mitigate the problem of potential network outages, certain PE container images are pre-stored on the dedicated nodes is the cluster. This cache only represents a meaningful subset of the overall PE container images, that is updated regularly.

IV. Preliminary Evaluation

To demonstrate the general feasibility of our approach, we implemented the presented location monitoring pipeline as well as the basic functionality of a lightweight node controller, that invokes relocation actions (offloading and onloading) on corresponding PE container, that is affected by changes in the geo-context of a delivery robot.

A. Sample dataset

The data used for our evaluation was gathered with *SensorLog*[4] on three tours (tour t_1, t_2 and t_3) in the city of Karlsruhe. While the first two tours stayed in the defined boundary, the third one violated the criteria at a specific point in time, which is healed later by re-entering again, as shown in Figure 5. The sample dataset contains a rich collection of features that also include the actual GPS data in WGS84 format as shown in Table I. The frequency at which the dataset was collected was set to 1Hz.

TABLE I
Excerpt of collected data for tour t_3.

tstamp	id	longitude	latitude	speed	battery	...
11:56:09	r_1	8.4254...	49.0120...	1.64	0.93	...
11:56:10	r_1	8.4255...	49.01200...	1.16	0.93	...
...	...	...	...	...	...	...
12:44:10	r_1	8.4252...	49.0121...	1.47	0.87	...

We encountered outages in the GPS signal, that was especially the case in t_3. For our evaluation, we disregard these missing data points. The tours vary in terms collected data points and duration, e.g., the tour t_3 took roughly 50 minutes at 419 data points. However, due to an interruption of the data collection process, where no data could be sensed, the data points only represent approximately 8 minutes worth of data. For the tests, only data of tour t_3 is used to evaluate our approach.

[4]http://sensorlog.berndthomas.net/ [last accessed: 18 March, 2019]

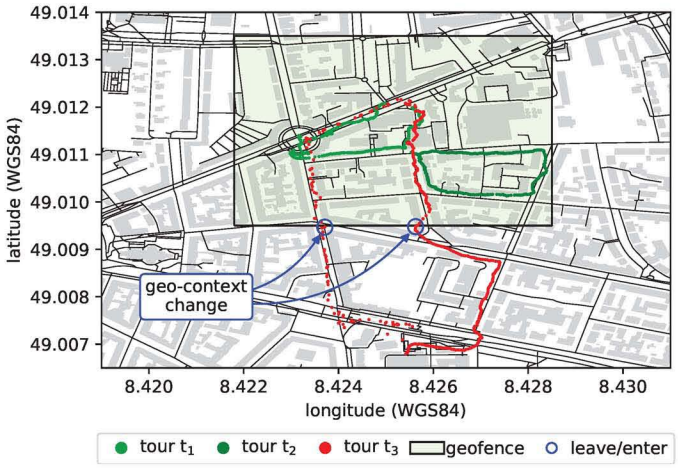


Fig. 5. Sample tours of GPS data with collected with *SensorLog*.

B. Implementation and Experimental Setup

The location monitoring pipeline SPP_{loc} consists of one PE source, that publishes a stream of GPS data of tour t_3 in WGS84 format and one PE processor, that subscribes to it and checks, whether the current location is contained in the given geofence geometry. Further, the result is published as a boolean value (see Definition 1).

Def. 1 (Location Monitoring Pipeline SPP_{loc})
The SPP_{loc} consists of two PE, where:

- $\text{PE}_{\text{S,gps}}$ is a PE source publishing a GPS data stream
- $\text{PE}_{\text{P,pip}}$ is a PE processor containing a point in polygon algorithm

For the PE processor, we used Shapely[5] to run the point in polygon algorithm. The two PE communicate via MQTT, which is realized with Eclipse Mosquitto[6]. Additionally, we packaged all PE as Docker images for the ease of deployment. Thus, we implemented a basic node controller, that leverages Dockers' remote API feature to access the Docker daemon. This allows us to remotely invoke relocation actions (offloading/onloading) on designated PE containers. We implemented the corresponding methods in the node controller, that are triggered for a change in the geo-context, e.g., leaving/entering the geofence. We deployed the pipeline on two local virtual machines: node 1 (edge) with 2 cores, 1GB memory, 10GB disk and node 2 (intermediary) with 2 cores, 2GB memory, 10GB disk, both running Ubuntu 16.04 and Docker 18.06. To account for this setup, we used Pumba[7], a network emulation tool especially for Docker containers and emulated a network delay of 1 sec between the two nodes. In the beginning, all PE containers are running on the edge node. In addition, we deployed an instance of the Mosquitto message broker on each node; a local MQTT edge broker and a remote MQTT intermediary broker.

[5]https://github.com/Toblerity/Shapely [last accessed: 18 March, 2019]
[6]https://mosquitto.org/ [last accessed: 18 March, 2019]
[7]https://github.com/alexei-led/pumba [last accessed: 18 March, 2019]

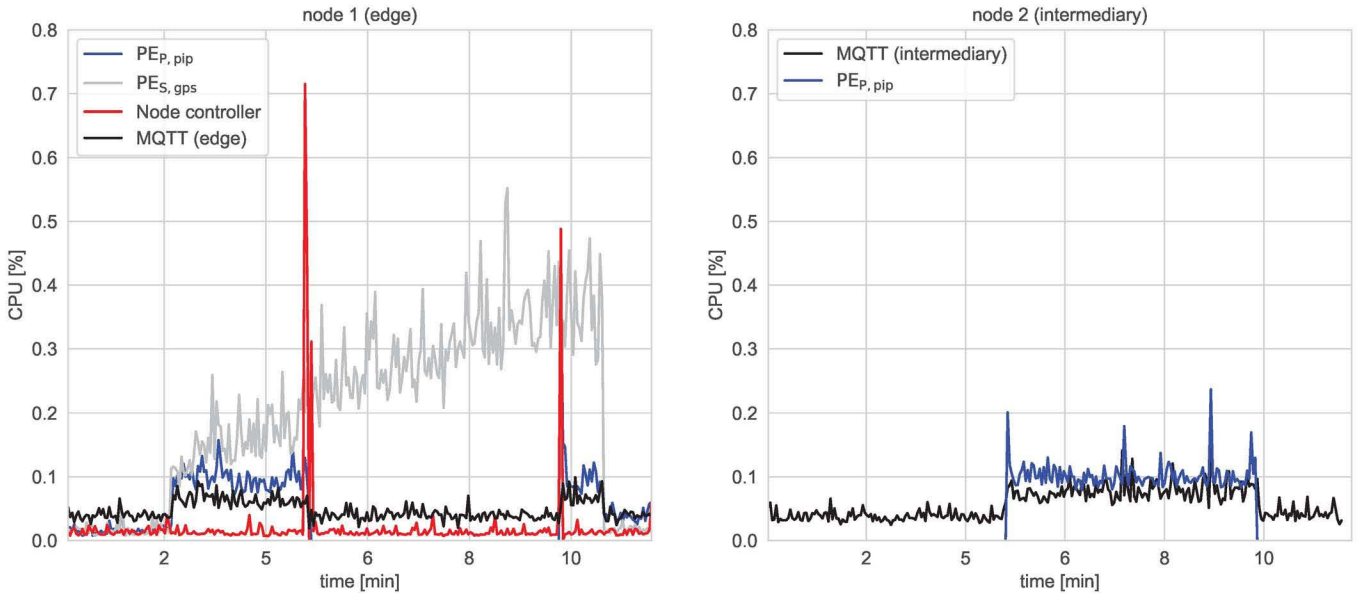


Fig. 6. CPU usage of Docker containers during the execution of the location monitoring pipeline SPP_{loc}. The node controller runs on node 1 (left).

a) Offloading: When the geofence is left in tour t_3 (geo-context change), the node controller immediately relocates the $_{P,pip}$ container to the intermediary node, while the notifying the $PE_{S,gps}$ container to now publish the sensor data to the intermediary MQTT instance. On the edge node, the existing PE processor container is removed, to free up resources.

b) Onloading: When the geofence is entered in tour t_3 (geo-context change), the node controller invokes the onloading of the $PE_{P,pip}$ container to establish the initial deployment setup, thus again notifying the $PE_{S,gps}$ container to publish the data to the edge MQTT instance. On the intermediary node, the existing PE processor container is removed.

In order to evaluate the footprint of our approach, we collect container runtime statistics during the pipeline execution phase for the sample data of tour t_3, that also includes relocating the PE processor container $PE_{P,pip}$ to the intermediary node and back on the edge node. However, we focus on the CPU usage per container in the following.

C. Results

The results of our prototypical implementation are shown in Figure 6. In the initial phase, all PE containers are running on node 1 (edge). For around 2 minutes, the PE containers idle at a base load of $< 0.1\%$ of CPU usage. After that, the PE source container $PE_{S,gps}$ starts publishing the GPS data stream of tour t_3 to the local MQTT (edge) broker. Instantly, the PE processor container $PE_{P,pip}$ starts to perform the point in polygon algorithm. Consequently, we observe an increase in its CPU usage, oscillating around 0.1%, while the node controller container still idles, waiting for geo-context changes. At around 5.5 minutes, we reach the location, where we left the geofence during the tour t_3. The node controller immediately invokes the offloading of the PE processor container $PE_{P,pip}$ to node 2 (intermediary), reconfigures the PE source container $PE_{S,gps}$ to publish the GPS data stream to the MQTT (intermediary) broker and kills the running PE processor container instance on node 1 (edge). This procedure reflects in a temporary peak of 0.71% of CPU usage by the node controller container. Shortly after the relocation is initiated, a new instance of the PE processor container $PE_{P,pip}$ spins up on node 2 (intermediary), catching up where the old instance stopped, by subscribing to dedicated topic on the MQTT (intermediary) broker, as depicted in Figure 6 (right). It is noticeable, that both the node controller container as well as the MQTT (edge) broker container on node 1 are back in idle mode. Thus, only consuming minimal resources. At around 9.8 minutes, we reach the location, where we enter the geofence during the tour t_3 again. Hence, the node controller invokes the onload action, thereby terminating the running instance on node 2 (intermediary) as well as notifying the PE source container $PE_{S,gps}$ on node 1 (edge) to publish to the MQTT (edge) broker. This onloading procedure leads to a temporary rise of CPU usage of the node controller container up to 0.48%. We further witness a steady increase in CPU usage of the PE source container $PE_{S,gps}$ during the whole execution time. This is mostly due to the fact, that the data is streamed from a CSV-file to the MQTT instances, while keeping the latest row pointer position in cache, thereby traversing over the rows. In a real streaming setup, we assume this to remain more stable. Therefore, the tests need to be further evaluated in a realistic setup to retrieve more reliable results. However, these preliminary evaluation shows, that the overall footprint is almost neglectable, with only minimal peaks of the node controller container, which is a good indicator due to the generally resource-limited edge nodes in typical FCI.

V. Related Work

In [2] a distributed dataflow programming paradigm is introduced and implemented by extending Node-RED[8]. Thereby, processing pipelines are executed on FCI based on static node capabilities and defined constraints. In [6] Kubernetes' label feature is used to provide static node metadata, e.g., hardware resources, or location, in order to deploy containers on suitable nodes. Foggy [10], a framework for dynamic resource provisioning and automated container-based application deployment is proposed, presenting a more fine-grained description of requirements including priority and privacy properties. In [11] IndieFog is presented and mentions various challenges for FCI, such as relying on a global federated adaptive service registry, that helps clients discover the best nodes for their applications, which involves both context-awareness and semantic descriptions of servers in terms of both spatio-temporal context and resource availability. However, this does not take SPP into consideration and is still only planned work. In [12] the FogFlow framework is introduced, that enables easy programming of elastic IoT services and supports standard interfaces for contextual data transfers across services in smart city scenarios. A container-based architecture for supporting autonomic data stream processing applications on FCI is shown in [13], yet not taken context changes into consideration. FogFrame [14] is another framework, that provides communication mechanisms for instantiating and maintaining service execution in the fog, thereby adapting to changes in available resources or workloads at runtime. The closest related work is [15], where a context-aware software framework for management of resources and service provisioning based on topology changes in FCI is proposed, mainly focusing on failover and handover management. In addition, in [16] a programming infrastructure for the geo-distributed computational continuum is presented, that manages the application components in a situation-aware manner.

Overall, the abovementioned approaches do not account for changes in the environment other than infrastructure related (e.g., node failures). Additionally, they neglect relevant dependencies between PE as well as the specifications of SPP, and do not allow for semantic reasoning to infer further knowledge.

VI. Conclusion and Future Work

Context-awareness in fog computing infrastructures is crucial, especially in scenarios, where edge nodes are mobile. This introduces several new challenges when managing dataflow based stream processing pipelines. In this paper, we presented our initial work on a conceptual architecture, that accounts for the context changes by introducing the node controller component, that invokes relocation actions on dedicated processing elements, when context changes are detected. Furthermore, we showed preliminary results of a prototypical implementation.

In our future work, we plan to further refine the architecture as well as the node controller component by leveraging existing semantic ontologies such as the SOSA ontology [17].

[8]https://nodered.org/ [last accessed: 18 March, 2019]

Acknowledgment

This research has been funded by the Federal Ministry for Economic Affairs and Energy of Germany (project number 01ME17008D).

References

[1] D. Riemer, L. Stojanovic, and N. Stojanovic, "SEPP: Semantics-based management of fast data streams," in *2014 IEEE 7th Int. Conf. on Service-Oriented Computing and Applications*, Nov 2014, pp. 113–118.

[2] N. K. Giang, M. Blackstock, R. Lea, and V. C. Leung, "Developing IoT applications in the Fog: A Distributed Dataflow approach," in *2015 5th Int. Conf. on the Internet of Things (IOT)*, Oct 2015, pp. 155–162.

[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and Its Role in the Internet of Things," *Proc. of the first edition of the MCC workshop on Mobile cloud computing*, pp. 13–16, 2012.

[4] B. I. Ismail, E. Mostajeran Goortani, M. B. Ab Karim, W. Ming Tat, S. Setapa, J. Y. Luke, and O. Hong Hoe, "Evaluation of Docker as Edge computing platform," in *2015 IEEE Conf. on Open Systems (ICOS)*, Aug 2016, pp. 130–135.

[5] P. Bellavista and A. Zanni, "Feasibility of Fog Computing Deployment based on Docker Containerization over RaspberryPi," in *Proc. of the 18th Int. Conf. on Distributed Computing and Networking - ICDCN '17*, 2017, pp. 1–10.

[6] C. Wöbker, A. Seitz, H. Mueller, and B. Bruegge, "Fogernetes: Deployment and management of fog computing applications," in *IEEE/IFIP Network Operations and Management Symposium: Cognitive Management in a Cyber World, NOMS 2018*, Apr 2018, pp. 1–7.

[7] D. Bermbach, F. Pallas, D. Garc, R. Kat, and S. Tai, "A Research Perspective on Fog Computing," *Research Paper*, 2017.

[8] E. Gamma, R. Helm, J. Ralph, and J. Vlissides, *Design Patterns : Element of Reusable Object Oriented Software*, 1995.

[9] P. Wiener, "Dynamic Management of Distributed Stream Processing Pipelines in Fog Computing Infrastructures," in *Doctoral Symposium of the 19th Int. Middleware Conf.*, Rennes, Britanny, France, 2018, [accepted for presentation].

[10] E. Yigitoglu, M. Mohamed, L. Liu, and H. Ludwig, "Foggy: A Framework for Continuous Automated IoT Application Deployment in Fog Computing," in *Proc. - 2017 IEEE 6th Int. Conf. on AI and Mobile Services, AIMS 2017*, 2017, pp. 38–45.

[11] C. Chang, S. N. Srirama, and R. Buyya, "Indie Fog: An Efficient Fog-Computing Infrastructure for the Internet of Things," *Computer*, vol. 50, no. 9, pp. 92–98, 2017.

[12] B. Cheng, G. Solmaz, F. Cirillo, E. Kovacs, K. Terasawa, and A. Kitazawa, "FogFlow: Easy Programming of IoT Services Over Cloud and Edges for Smart Cities," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 696–707, Apr 2018.

[13] A. Brogi, G. Mencagli, D. Neri, J. Soldani, and M. Torquati, "Container-Based Support for Autonomic Data Stream Processing Through the Fog," *Euro-Par 2017: Parallel Processing Workshops*, pp. 17–28, 2018.

[14] O. Skarlat, V. Karagiannis, T. Rausch, K. Bachmann, and S. Schulte, "A Framework for Optimization, Service Placement, and Runtime Operation in the Fog," in *2018 IEEE/ACM 11th Int. Conf. on Utility and Cloud Computing (UCC)*. IEEE, Dec 2018, pp. 164–173.

[15] S. Pešić, M. Tošić, O. Iković, M. Ivanović, M. Radovanović, and D. Bošković, "Context aware resource and service provisioning management in fog computing systems," *Studies in Computational Intelligence*, vol. 737, pp. 213–223, Oct 2017.

[16] E. Saurez, K. Hong, D. Lillethun, U. Ramachandran, and B. Ottenwälder, "Incremental deployment and migration of geo-distributed situation awareness applications in the fog," in *Proc. of the 10th ACM Int. Conf. on Distributed and Event-based Systems - DEBS '16*. New York, New York, USA: ACM, 2016, pp. 258–269.

[17] K. Janowicz, A. Haller, S. J. Cox, D. L. Phuoc, and M. Lefrançois, "SOSA: A lightweight ontology for sensors, observations, samples, and actuators," *Journal of Web Semantics*, 2018.