



HAL
open science

A federated fog-cloud framework for data processing and orchestration: A Case Study in Smart Cities

Dapeng Lan, Yu Liu, Amir Taherkordi, Frank Eliassen, Stéphane Delbruel,
Liu Lei

► To cite this version:

Dapeng Lan, Yu Liu, Amir Taherkordi, Frank Eliassen, Stéphane Delbruel, et al.. A federated fog-cloud framework for data processing and orchestration: A Case Study in Smart Cities. SAC '21: The 36th ACM/SIGAPP Symposium on Applied Computing, Mar 2021, Virtual Event Republic of Korea, South Korea. pp.729 - 736, 10.1145/3412841.3444962 . hal-04901341

HAL Id: hal-04901341

<https://hal.science/hal-04901341v1>

Submitted on 24 Jan 2025

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Federated Fog-Cloud Framework for Data Processing and Orchestration: A Case Study in Smart Cities

Dapeng Lan
Department of Informatics
University of Oslo
Oslo, Norway
dapengl@ifi.uio.no

Yu Liu
Department of Science and
Technology
Linköping University
Linköping, Sweden
yu.a.liu@liu.se

Amir Taherkordi
Department of Informatics
University of Oslo
Oslo, Norway
amirhost@ifi.uio.no

Frank Eliassen
Department of Informatics
University of Oslo
Oslo, Norway
frank@ifi.uio.no

Stéphane Delbruel
Department of Informatics
University of Oslo
Oslo, Norway
stephde@ifi.uio.no

Lei, Liu
Xidian University
Xi'an, China
tianjiaoliulei@163.com

ABSTRACT

The fog computing paradigm has been proposed to alleviate the pressures on cloud platforms for data processing and enable computation-intensive and delay-sensitive applications in smart cities. However, state-of-the-art approaches mainly advocate either cloud- or fog-based data processing solutions, and they also lack a common framework for programming over the fog-cloud continuum. In this paper, we propose a distributed, fog-cloud data processing and orchestration framework, which is capable of exploiting the semantics of both fog platforms and the Cloud. Our framework can create on-demand process engine data flow (PEDF) spanning multiple device layers with various resource constraints. This will considerably help the developers rapidly develop and deploy data processing applications over the fog-cloud continuum. Our proposed framework is validated in a real-world scenario—IoT data streaming analytics for the smart green wall in a smart city—which demonstrates efficient resource usage and latency reduction.

CCS CONCEPTS

• **Computer systems organization** → **n-tier architectures.**

KEYWORDS

Fog computing, Data processing, Cloud computing, Smart City, Orchestration

ACM Reference Format:

Dapeng Lan, Yu Liu, Amir Taherkordi, Frank Eliassen, Stéphane Delbruel, and Lei, Liu. 2021. A Federated Fog-Cloud Framework for Data Processing and Orchestration: A Case Study in Smart Cities. In *The 36th ACM/SIGAPP Symposium on Applied Computing (SAC '21), March 22–26, 2021, Virtual*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SAC '21, March 22–26, 2021, Virtual Event, Republic of Korea

© 2021 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-8104-8/21/03...\$15.00

<https://doi.org/10.1145/3412841.3444962>

Event, Republic of Korea. ACM, New York, NY, USA, 8 pages. <https://doi.org/10.1145/3412841.3444962>

1 INTRODUCTION

The term Internet of Things (IoT) was proposed to reflect connectivity between the physical and the digital world over the Internet. The research on IoT applications has been evolving in various domains, of which smart city is a promising representative. IoT technologies in smart cities offer excellent opportunities to both public and private sectors to harvest data sources, obtain data insights, and make data-driven decisions [1, 2]. Conventional smart city services are typically centralized, i.e., data storage, data analytics, and feedback control solely rely on remote servers in the Cloud. However, the cloud approach is difficult to fulfill the new services requiring high throughput and low latency [3], such as augmented reality (AR) and autonomous vehicles [4, 5]. Therefore, the fog and edge computing paradigms have been proposed to address the above challenges by moving parts of processing tasks from the Cloud to the edge network devices, closer to the data source [6].

The adoption of fog computing frameworks for data processing in smart cities introduces various challenges: 1) different from cloud computing, fog computing nodes are more heterogeneous in terms of hardware resources (e.g., different CPU architectures with x86, arm32 or arm64), 2) due to a large number of fog computing nodes, the computing framework requires a proper abstraction layer to manage these nodes, and 3) due to the heterogeneity of fog computing nodes, it is more complicated to develop and deploy software applications and arrange the execution process for the application. Furthermore, it is essential to have a hybrid approach, combining cloud and fog data processing, to fully utilize the resources. For instance, heavy computing tasks with latency-tolerant requirements can be processed in the Cloud while latency-sensitive tasks should be deployed on the fog nodes.

There exist a number of frameworks and engines for data processing and analytics within the Cloud, such as Apache Spark [7] and Apache Flink [8]. In these frameworks, edge and end devices are used only as proxies to transmit data to the Cloud. Several fog/edge data processing engines have been introduced lately (e.g., Apache

Edgent[9] and Apache Minifi[10]). These engines usually process local raw data on resource constraint IoT devices, which reduces the load on the Cloud with respect to data processing and storage. Hybrid approaches, combining both cloud and fog processing, require deploying two different process engines. This introduces two types of challenges considering programming in such settings: how to address synchronization overheads between the two engines and how to split the data processing workflow into two platforms. Moreover, application developers lack abstraction layers and need to spend extra time and resources to consider both computing platforms and frameworks.

We propose a distributed, fog-cloud data processing and orchestration framework leveraging the data-flow programming model to tackle the above challenges. This framework provides a conceptual structure and facilitates the fog-cloud data processing application development dealing with heterogeneous infrastructures. We implement and experimentally evaluate our framework in a real-world scenario—a smart green wall in the smart city. Our contributions can be summarized as follows:

- (1) We propose a conceptual fog-cloud system architecture for dealing with the heterogeneity of the hardware resources among devices.
- (2) We propose a distributed, fog-cloud data processing and orchestration framework leveraging the data-flow programming model, alleviating the complexity for developers to rapidly develop and deploy data processing applications.
- (3) We implement the proposed fog-cloud data processing framework and experimentally evaluate it with a representative real-world scenario which is a smart green wall in smart cities for IoT data stream analytics.

The rest of the paper is organized as follows. In Section 2, we discuss the related work. In Section 3, we present the proposed system architecture and the orchestration framework for a smart city and then we explain the system implementation in Section 4. In Section 5, we evaluate our framework through a real use case, a smart green wall application. We conclude the paper and discuss the future work in Section 6.

2 RELATED WORK

In this section, we split the related work into two major branches: cloud data processing platforms in smart cities and fog/edge-cloud based data processing and orchestration frameworks.

Cloud platforms, with elastic computation, storage, and network resources, have been popularly used in the past years, processing large amounts of IoT data in smart cities. The cloud data processing platforms are quite mature nowadays. For instance, the Hadoop MapReduce framework [11] and Spark [7] have been vastly used to perform large-scale batch processing tasks. Stream processing frameworks are also proposed to reduce the time that data streams spend in the data pipeline, such as Storm, Samza and Flink [8]. More recently, some software frameworks (e.g., Apache Edgent[9] and Apache Minifi[10]) have been introduced for fog/edge devices that have constrained computing and storage resources. However, the above frameworks mainly focus on either cloud- or fog-based solutions and are still far away from a general data process framework working across cloud and fog platforms.

There are also some research works combining the Cloud and fog for data processing and orchestration. Hong *et al.* [12] proposed MobileFog, which is a high-level programming model for IoT with geospatially distributed and latency-sensitive properties. It provides a general tree structure to map services to devices. In the work [13], Giang *et al.* introduced a distributed data flow (DDF) programming model by extending Node-RED, as a basis for fog-based IoT applications. DDF provides an easy and flexible way to design and develop IoT applications. B. Cheng *et al.* introduced the FogFlow framework in [14] for cloud and edge platforms. FogFlow extends the dataflow programming model and utilizes the *NGSI* standard, enabling easy programming of elastic IoT services. However, the above architectures are not general frameworks as they need specific interfaces. Yangui *et al.* [15] presented a PaaS architecture aiming for hybrid cloud and fog environments by utilizing Cloud Foundry and REST communication interfaces. However, in the above work, applications cannot run standalone in fog and need support from the Cloud. In [16], a high-level fog framework is designed, called Fog05, to provision, manage and monitor the network that spans across things, edge and the Cloud. However, it is still in an infant stage and needs to specify plugin, Zenoh, to support the system.

Apart from the academic community, several frameworks for fog-cloud data processing and orchestration have been proposed by the industries, with tools for application development and deployment, such as the Cisco edge computing framework, Amazon Greengrass, Microsoft Azure IoT, Google CloudIoT, Foghorn [17], and Nebbiolo [18]. There are also some open-source projects or frameworks dealing with fog-cloud data processing and orchestration, such as ioFog [19], BAETYL [20] and EdgeX Foundry [21].

Despite the above works combining the Cloud and fog computing, these systems use different standards and programming approaches and need specific interfaces or specific plugins. Therefore, in this study, we propose a fog and cloud data processing and orchestration framework, as a supplementary enhancement and feasibility verification to the state of the art in federated fog-cloud computing. This study is not intended to compete with aforementioned frameworks that are put into industrial and academic practice, but rather provided as a real implementation and validation for the federated fog-cloud framework for data processing and orchestration in smart cities, contributing valuable inputs to the related research fields.

3 ARCHITECTURE AND ORCHESTRATION FRAMEWORK

This section presents an overview of the proposed system architecture and orchestration framework for a smart city as shown in Fig. 1. We then explain the details of the process engine orchestration framework based on a data-flow programming model and its components. In the end, we compare different techniques for data processing in the Cloud, fog and end devices.

3.1 System Architecture

As shown in Fig. 1, the system architecture consists of three layers: the Cloud layer, the Fog layer, and the End Devices (ED) layer. The ED layer contains various types of smart devices (e.g., mobile

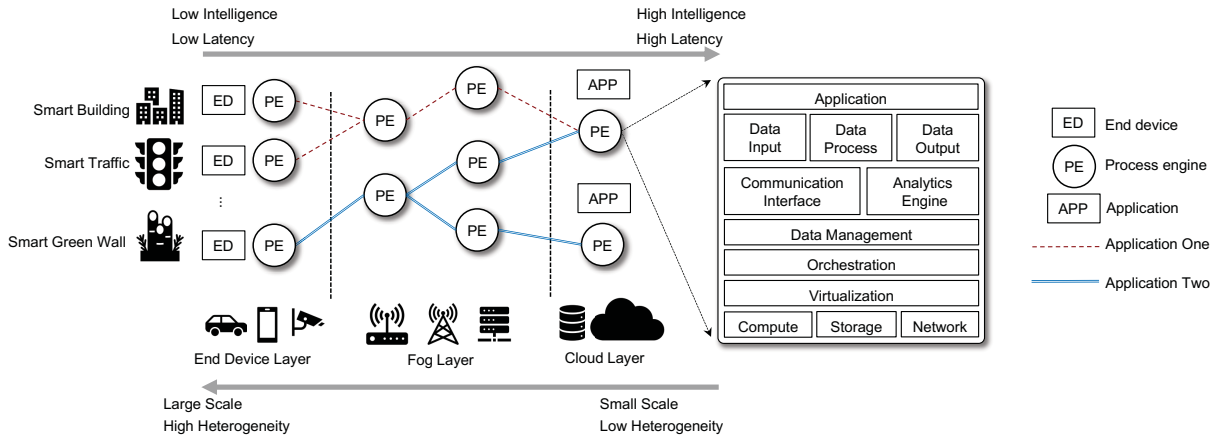


Figure 1: A federated fog-cloud framework for data processing and orchestration in smart cities

phones, wearable equipment, and intelligent vehicles) with limited computational and storage resources. The end devices generate various types of data for different applications, such as smart building, smart traffic, and smart green wall. The *Fog layer* consists of various fog nodes communicating with different kinds of ED. They can serve as switches at home or as base stations. A fog node resides at the network edge, serving at the intermediate layer between the ED layer and the Cloud layer. The *Cloud layer* is for processing services with high computational and storage resources demands.

Smart city applications come with the properties of large scale IoT data, high heterogeneity of devices, distributed intelligence, and various quality of service requirements [22]. From the Cloud layer to the ED layer, the data scale is dramatically increasing, and the heterogeneity of the devices also becomes high. Latency is much lower at the ED layer as the tasks run closer to the data sources. However, from the Cloud layer to the ED layer, the intelligence decreases because the ED layer lacks high computational capacity. Due to heterogeneity of computing platforms from the Cloud to Fog and ED layers, we design the process engine orchestration framework (PEOF) utilizing the data-flow programming model. This design principle creates a data-flow pipeline for applications across different layers, which hides the complexities for application developers as they do not need to consider the heterogeneity of underlying devices. As shown in Fig. 1, applications One and Two are represented by the process engine data flow across the ED, Fog and Cloud layers.

We next describe the process engine orchestration framework with more details (components and functions), as well as their deployment.

3.2 Process engine orchestration framework

3.2.1 Process Engine. The process engine (PE) can be regarded as an operator distributed in different layers, which is also *platform-agnostic*. Fig. 1 shows the main components of PE, consisting of hardware, virtualization, orchestration, data management, analytics engine, communication interfaces, and data and application layers. Fig. 2 shows techniques and tools that can be applied in the process engine components for the Cloud, Fog and End Device layers.

Computation Layer	Cloud	Fog	End Device
Application	Complex Event Processing, Streaming/Batching, Visualization and Monitoring	IoT Streaming/Batching, Context-Aware Services, Industrial Control	Real-time Crowdsensing, Data Fetching and Filtering
Data Analytic Engine	Spark, Storm, Flink, Beam	Spark, Storm, Flink, Beam, minifi, Edgent	Simple Local Analytic, Minifi, Edgent
Data Management	Kafka, Mongo DB, Cassandra, NoSQL	MySQL, SQLite	SQLite, Document Files
Communication Interface	HTTP, Kafka	HTTP, Kafka, MQTT	MQTT, CoAP
Orchestration	YARN, MESOS, Kubernetes	Kubernetes, KubeEdge, Docker Swarm	KubeEdge, K3s
Virtualization	Virtual Machine, Container	Virtual Machine, Container	Container, Unikernel
Hardware	Openstack, SDN, NFV Computation, Storage, Network		

Figure 2: Different techniques and tools for data processing in the Cloud, Fog and End Devices layers

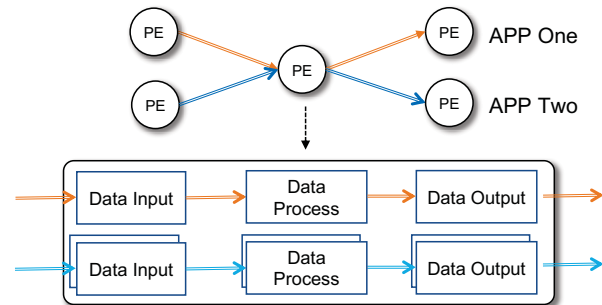


Figure 3: Process Engine with data flow programming model

- **Hardware layer:** The heterogeneous devices have various kinds of resources (computation, storage and network) and corresponding operating systems (OS). Which hardware to choose depends on the target application scenario and service requirements.
- **Virtualization layer:** This layer addresses the configuration and virtualization of the system hardware resources. Virtualization plays an important role in service development and service integration due to heterogeneity of devices.

Nowadays, the popular virtualization techniques are Openstack, Software-Defined Network (SDN), Network Function Virtualization (NFV), Virtual Machines (VM), containers, Unikernel, etc. For instance, in our application scenario, we utilize Docker to modularize the applications so that the applications become portable between multiple platforms.

- **Orchestration layer:** This layer is responsible for deploying services on the corresponding devices. The popular orchestration techniques are Yarn, Mesos and Kubernetes for the Cloud layer, and Kubernetes, KubeEdge and K3S for the Fog and ED layers.
- **Data management layer:** This layer addresses data storage and distribution, including file systems, databases, caches, and data lakes. There are various data management frameworks, such as Kafka, Cassandra, MySQL, and SQLite. These data management frameworks are usually used together with different communication interfaces and analytics engines.
- **Analytics engine layer:** This layer deploys the application-specific processing framework for running the tasks and jobs, which are usually divided into two categories: batch jobs and streaming jobs. Recently, many data processing and analytics frameworks have been proposed such as Apache Spark, Flink, Storm, Beam, Edgent, Nifi, and Minifi. Moreover, different machine learning (ML) libraries have been developed based on these frameworks, such as Tensorflow and Pytorch.
- **Communication interface layer:** This layer is responsible for message communication and data exchange, usually equipped with publish-subscribe messaging systems. The popular communication interfaces are HTTP, Kafka, MQTT, and CoAP.
- **Applications layer:** This layer defines the objective and logic of data processing jobs. For instance, the Cloud layer usually deals with complex event processing, streaming and batching applications, and visualization and monitoring applications. The Fog layer commonly processes IoT streaming and batching applications, context-awareness services, and industrial control. End devices normally run tasks from real-time crowdsensing, data fetching and filtering.

3.2.2 *Process Engine Data Flow.* Fig. 3 shows the deployment of process engine data flow (PEDF) for applications One and Two. PEDF is a directed acyclic graph (DAG) of PEs and each application has its own PEDF. Each PE mainly contains three kinds of processing modules: *Data Input*, *Data Process* and *Data Output*. The PEDF for application Two has multiple modules in each PE because each operator may have multiple data inputs, data process tasks and data outputs. The *Data Input* module contains the communication interfaces, which are the connectors among different PEs. It can be sensor readings from end devices or data from another PE. *Data Process* is the main module in PE for processing the data, which is supported by various data analytics engines and data management systems. *Data Process* is a self-contained atomic process consisting of the application logic that can run standalone. Data Process can also interact with the local storage and exchange messages. *Data Output* module also contains the communication interfaces for sending the data to other PE(s) and connecting to the next PE.

Table 1: Testbed parameters and performance metrics

	Cloud node	Fog node	ED node
OS	Ubuntu 18.04	Raspbian	Raspbian
CPU	1(Intel 2.4Ghz)	4(Cortex 1.4Ghz)	4 * 1.4Ghz
RAM	4 GB	1GB	1GB
DISK	256GB	64GB	64GB
Hardware	NREC	Raspberry Pi 3 B+	Pi 3 B+

4 SYSTEM IMPLEMENTATION

As Fig. 3 shows, by connecting the PEs from different layers, these PEs form the execution process for the application. When designing these PEs, there are some principles needed to be considered. These design principles can be adjusted depending on the system requirements.

- **Independency:** As shown in Fig. 3, one device can run multiple processes for different applications. Therefore, these processes need to be independent without interfering with each other. Virtualization techniques such as VM and containers, provide good support for running processes independently.
- **Priority:** Resources on devices are limited, thereby mechanisms for determining priority should be considered for each process. The priority is defined by the developers when deploying the services.
- **Standard:** As there are various techniques and tools for data processing, shown in Fig. 2, standards are needed for the integration of these components. For instance, from the communication interface perspective, communication protocols and data exchange formats should be standardized like MQTT or Kafka. Which standard to choose also depends on the application scenario and the industrial domain.

PEDF is flexible and can be adapted to different use-cases and application scenarios. The execution process can be represented as a directed acyclic/cyclic graph. In the next section, we will show how to implement the data processing and orchestration framework.

In this section, we focus on the detailed implementation of the data process and orchestration framework. The core of our proposed data processing framework is to exploit the semantics of both the Cloud and fog platforms in order to provide a user-friendly orchestration framework. Table 1 shows the testbed parameters and performance metrics. We use one Raspberry Pi 3 B+ as our end devices and fog nodes with Raspbian operating system, and one Intel 2.4Ghz CPU machine in NREC (A Norwegian Cloud Infrastructure for Research and Education)[23] cloud with the Ubuntu 18.04 operating system. Fig. 4 shows the deployment of hardware devices in a green plant wall system as our use case. The Raspberry Pi equipped with various kinds of sensors (such as temperature, humidity, and light sensors) is mounted at the top of the green plant wall. Fig. 5 shows the technical implementation for process engines and Fig. 6 depicts the platform setup.

Communication interface: Message Queue Telemetry Transport (MQTT) is a lightweight publish-subscribe protocol that is popularly used in IoT fields. We use the open-source Mosquitto MQTT broker as our message handler in end devices. Apache Kafka is



Figure 4: Deployment to a green plant wall system.
















Computation Layer	Cloud	Fog	End device
Application	Grafana 	IoT Streaming/Batching Signal Control	Real-time Crowdsensing, Data Filtering
Data Analytic Engine	Flink 	Flink 	Python 
Data Management	Cassandra 	MySQL 	JSON 
Communication Interface	Kafka 	Kafka, MQTT 	MQTT 
Orchestration	Kubernetes 	Kubernetes 	
Virtualization	Docker 	Docker 	
	Openstack 		

Figure 5: Proposed system implementation for smart green wall case study

an open-source stream-processing software platform. Kafka provides a unified, high-throughput, low-latency platform for handling real-time data feeds but more resource-hungry.

Containerization and orchestration: Container orchestration benefits deploying fog-cloud applications at scale. Container is a standardized way to package software components and their dependencies, which helps these applications to be easily redeployed. We use Docker as our container running environment, and we use Kubernetes as our container orchestrator. Kubernetes is an open-source container orchestration system for automating application deployment, scaling, and management. Nowadays, Kubernetes is commonly used in public cloud platforms.

Analytics engine: Our data analytics engine aims to support event-driven and data-flow applications. We use open-source Apache Flink as our data analytics engine for cloud and fog nodes. Apache Flink is a distributed stream processor, which also supports batch

processing. Apache Flink has intuitive and expressive interfaces to enable various stateful/stateless data processing services.

Development and deployment: We use Docker's Buildx [24] feature to create customized Docker containers for our components, such as Apache Flink, MQTT and Kafka. After that, we use Kubernetes to compose the services running on the cluster crossing the Cloud and fog nodes. We wrap the components by a YAML script, which defines where the service is deployed, how much resource is utilized by each service, and connection between the services and network configurations. Benefiting from the combination of Docker and Kubernetes, the developer can easily develop the application components without worrying about hardware layers.

Application running: For the end device nodes, we use the Mosquitto MQTT broker as our communication interface, JSON files as local storage and python programs as simple local analytics. The main functions of this process engine are real-time data reading and primitive data filtering, such as error data. Then, the PE in end devices will send the processed data to the PE in the fog nodes. The fog nodes can either store the data locally in the MySQL database or push the data to the Apache Flink for further processing. We can also use a machine learning library here for basic data inference to help make automatic decisions, such as watering the green wall. After that, the fog nodes will push the data to the Cloud using Kafka. Our cloud platform is built using the Openstack platform. After receiving the data from the fog nodes by Kafka, the Cloud pushes the data to the Apache Flink engine for data analytics and stores the data in the Cassandra database for long-term storage. The processed data can also be sent to the Grafana visualization platform or fetched by the third party APIs through the Cassandra database.

5 EVALUATION AND RESULTS

A smart green wall is a vertical wall with plants growing on the surface, which has a positive impact on improving people's living comfort and reducing indoor pollutants. The green wall system has watering, lighting, and ventilation systems to grow the plant healthily on the wall. Several temperatures, humidity and illumination sensors are installed in the system to monitor the changing of the indoor climate in order to provide feedback control. The hardware setup of the smart green wall is also partly illustrated in our previous work[25]. A smart green wall application is selected to verify the feasibility of our data processing and orchestration framework for smart cities and evaluate the performance.

Fig. 4 shows the deployment of the framework to a green wall system. Previously, the functioning of the green wall was solely relying on a public cloud-based monitoring and management solution [26], i.e., the sensed data from geographically distributed green wall applications is transmitted to the central Cloud where data processing, storage, and analytics take place. However, an explicit partitioning for data processing tasks between fog nodes and the Cloud is still missing. The system also faces the threat of single point of failure (SPOF) if it loses the communication connection to the Cloud. As a continuation of our previous work [26], we elevate the power of fog nodes by exploiting the proposed fog-cloud data processing and orchestration framework to the green wall. In this study, sensors keep measuring indoor environmental parameters

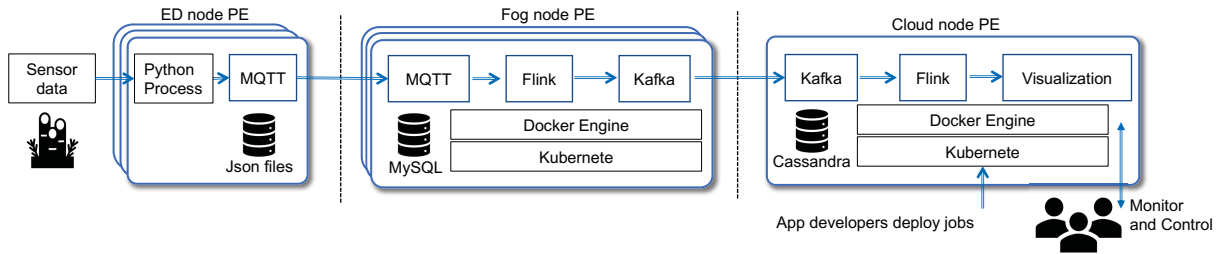


Figure 6: Proposed system setup for smart green wall



Figure 7: Visualization system for smart green wall system

and transmitting them to the fog node instead of the central Cloud. Light-weight data pre-processing is performed in the fog node to filter the invalid and trivial data and accelerate further data analytics. The monitoring and control functions are also shifted from the Cloud to the fog, which are implemented as a standalone application running on the fog device. For critical tasks such as watering that need timely feedback control in abnormal situations, the fog node can also make decision locally in time. After aggregation of sensor data from various sources, the fog node then sends the data to the Cloud for further processing and storing. Additionally, graphical visualization of sensor data also takes place in the Cloud, taking advantage of the powerful computing capability and the flexibility of multiple visualization frameworks in the Cloud.

A sample of the collected sensor data from a green wall utilizing the proposed framework is presented in Fig. 7 using the Grafana visualization tool [27]. This figure shows the data collected from the temperature, humidity, light, pm2.5 and CO₂ sensors. It depicts the CO₂ level in the environment with around 440-460 ppm during the evening. It also presents the temperature and humidity changes

during this period which can be clearly observed. Light sensors help track whether plants have enough light for photosynthesis. PM sensors can track air quality in the environment. The water consumption pattern can also be easily noticed. If the water level is lower than the pre-defined threshold, a red alarm is reported. Then the fog node can send the control signal to fill the water for that plant. It will not be efficient for human to monitor, predict and water a large number of green walls. Therefore, deploying a machine learning model on the fog nodes can support anomaly detection and automatic watering control, which reduces human labor intervention. The machine learning models can be trained in the Cloud and deployed to the fog nodes. Using different kinds of machine learning models for the green-wall application is not within the scope of this paper. The details can be referred to the work [28].

5.1 Machine learning performance comparison

This section further compares the performance on resource usage of machine learning tasks in fog and cloud platforms, as shown

Table 2: Resources comparison for Machine learning tasks in fog and cloud platforms

Hardware	Algorithm	Time		CPU load		Memory usage	
		Training	prediction	training	prediction	training	prediction
Cloud VM (1x Intel Processor 2.4GHz, 4G Memory)	LSTM-ED	252 s	12.24 ms	94,65%	98,74%	81,74%	81,55%
Raspberry Pi 3B+ (4x armv7 1.4GHz, 1G Memory)	LSTM-ED	2005 s	87.1 ms	35,46%	26,06%	90,28%	89,9%

in Table 2. In this performance comparison, we implemented an LSTM-ED algorithm that is used for anomaly detection of indoor climate in both cloud and fog nodes. The training time consumption in the Cloud is 252 seconds, which is much less than the training time (2005 seconds) cost on the edge node. The prediction time in the Cloud side is 12.24 milliseconds while in the fog side is 87.1 milliseconds. This is due to the fact that the Cloud benefits from more computational resources than the fog node. As we can see from Table 2, machine learning tasks take full usage of the computing resources in the cloud node, leading to above 90% CPU load of a single core processor in both training and prediction processes, while the fog node equipped with an embedded four-core processor can hardly shift more computing resources to improve efficiency. The cloud node utilizes near 81% of assigned 4G memory in both training and prediction processes while the fog node consumes near 90% of its 1G memory, which further enlarges the performance gap, let alone the computing resources in the Cloud can be flexibly scaled if necessary.

5.2 Latency performance comparison

In this section, we further compare the task performance between fog and cloud layers. To clearly show the advantage of using the fog computing framework, we have tested the processing time on the nodes and the round trip time (RTT) in different scenarios. Figure 8 shows the comparison of on-node processing time between fog and cloud nodes. The x-axis indicates the number of performed tests while the y-axis denotes the on-node processing time. The result shows that the on-node processing time in the fog is larger than that in the Cloud, as the cloud platform has more resources to process the tasks. Figure. 9 presents the comparison of round trip time between fog and cloud nodes. It shows that the average RTT time for the Cloud is near 65 ms, which is larger than in the fog side with near 33 ms. The reason is that fog nodes are closer to the field node as compared with that of the cloud platform. The above results show that using the federated fog-cloud platform can efficiently reduce the latency while improving the system performance. For instance, the machine learning model can be trained in the cloud platform to take full advantage of the computing resources and the trained model then can be transferred and deployed on the fog node to guarantee low-latency decision-making.

6 CONCLUSIONS

In this paper, we proposed a distributed, fog-cloud data processing and orchestration framework. This framework is capable of exploiting the semantics of both the Cloud and fog platforms, which creates on-demand process engine data flow (PEDF) spanning multiple devices with various resource constraints. We explained the details of the framework design (logic, components and functions) and framework implementation. Our real-world smart green wall

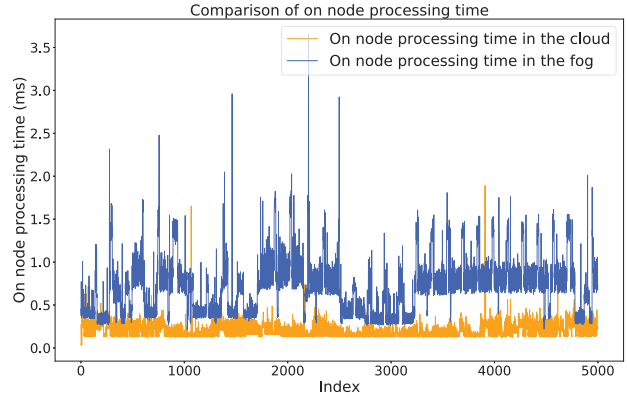


Figure 8: Comparison of on node processing time.

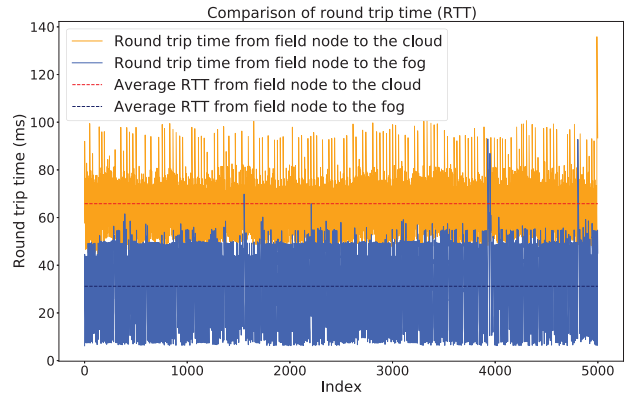


Figure 9: Comparison of round trip time.

application demonstrates the effectiveness of the orchestration framework integrating the Cloud and fog nodes. In the future, we will investigate dynamic service configurations in the fog-cloud data processing and orchestration framework.

ACKNOWLEDGMENT

This work was supported by the Norwegian Research Council under the DILUTE project (Grant No. 262854/F20), and it was also partly supported by the National Natural Science Foundation of China under Grant No. 62001357 and the Swedish Innovation Agency, Vinnova.

REFERENCES

- [1] Z. Lv, B. Hu, and H. Lv. Infrastructure monitoring and operation for smart cities based on iot system. *IEEE Transactions on Industrial Informatics*, 16(3):1957–1962, 2020.

- [2] Dapeng Lan, Zhibo Pang, Carlo Fischione, Yu Liu, Amir Taherkordi, and Frank Eliassen. Latency analysis of wireless networks for proximity services in smart home and building automation: The case of thread. *IEEE Access*, 7:4856–4867, 2018.
- [3] J. Yue, M. Xiao, and Z. Pang. Distributed fog computing based on batched sparse codes for industrial control. *IEEE Transactions on Industrial Informatics*, 14(10):4683–4691, Oct 2018.
- [4] Lei Liu, Chen Chen, Qingqi Pei, Sabita Maharjan, and Yan Zhang. Vehicular edge computing and networking: A survey. *Mobile Networks and Applications*, pages 1–24, 2020.
- [5] Lei Liu, Chen Chen, Tie Qiu, Mengyuan Zhang, Siyu Li, and Bin Zhou. A data dissemination scheme based on clustering and probabilistic broadcasting in vanets. *Vehicular Communications*, 13:78–88, 2018.
- [6] P. Mach and Z. Becvar. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys Tutorials*, 19(3):1628–1656, thirdquarter 2017.
- [7] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, HotCloud'10*, page 10, USA, 2010. USENIX Association.
- [8] Paris Carbone, Stephan Ewen, Gyula Fóra, Seif Haridi, Stefan Richter, and Kostas Tzoumas. State management in apache flink®: Consistent stateful distributed stream processing. *Proc. VLDB Endow.*, 10(12):1718–1729, August 2017.
- [9] Apache edgent. Accessed September 28, 2020. <http://edgent.apache.org>, 2020.
- [10] Apache minifi. Accessed September 28, 2020. <https://nifi.apache.org/minifi/>, 2020.
- [11] Jens Dittrich and Jorge-Arnulfo Quiané-Ruiz. Efficient big data processing in hadoop mapreduce. *Proceedings of the Vldb Endowment*, 5(12):2014–2015, 2012.
- [12] Kirak Hong, David Lillethun, Umakishore Ramachandran, Beate Ottenwälder, and Boris Koldehofe. Mobile fog: A programming model for large-scale applications on the internet of things. In *Proceedings of the Second ACM SIGCOMM Workshop on Mobile Cloud Computing, MCC '13*, page 15–20, New York, NY, USA, 2013. Association for Computing Machinery.
- [13] N. K. Giang, M. Blackstock, R. Lea, and V. C. M. Leung. Developing iot applications in the fog: A distributed dataflow approach. In *2015 5th International Conference on the Internet of Things (IOT)*, pages 155–162, 2015.
- [14] Bin Cheng, Gurkan Solmaz, Flavio Cirillo, Erno Kovacs, Kazuyuki Terasawa, and Atsushi Kitazawa. Fogflow: Easy programming of iot services over cloud and edges for smart cities. *IEEE Internet of Things Journal*, pages 1–1, 2017.
- [15] S. Yangui, P. Ravindran, O. Bibani, R. H. Glitho, N. Ben Hadj-Alouane, M. J. Morrow, and P. A. Polakos. A platform as-a-service for hybrid cloud/fog environments. In *2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, pages 1–7, 2016.
- [16] Angelo Corsaro and Gabriele Baldoni. fogØ5: Unifying the computing, networking and storage fabrics end-to-end. In *3rd IEEE Cloudification of Internet of Things*, 2018.
- [17] Official foghorn website. Accessed September 28, 2020. <https://www.foghorn.io/>, 2020.
- [18] Official nebbiolo website. Accessed September 28, 2020. <https://www.nebbiolo.tech/>, 2020.
- [19] Official iofog website. Accessed September 28, 2020. <https://iofog.org/>, 2020.
- [20] Official baetyl website. Accessed September 28, 2020. <https://baetyl.io/>, 2020.
- [21] Official edgex foundry website. Accessed September 28, 2020. <https://www.edgexfoundry.org/>, 2020.
- [22] Dapeng Lan, Amir Taherkordi, Frank Eliassen, and Geir Horn. A survey on fog programming: Concepts, state-of-the-art, and research challenges. In *Proceedings of the 2nd International Workshop on Distributed Fog Services Design, DFSD '19*, page 1–6, New York, NY, USA, 2019. Association for Computing Machinery.
- [23] Norwegian research and education cloud. Accessed September 28, 2020. <https://www.nrec.no/>, 2020.
- [24] Docker buildx. Accessed September 28, 2020. <https://docs.docker.com/engine/reference/commandline/buildx/>, 2020.
- [25] Y. Liu, K. Akram Hassan, M. Karlsson, O. Weister, and S. Gong. Active plant wall for green indoor climate based on cloud and internet of things. *IEEE Access*, 6:33631–33644, 2018.
- [26] Y. Liu, K. Akram Hassan, M. Karlsson, Z. Pang, and S. Gong. A data-centric internet of things framework based on azure cloud. *IEEE Access*, 7:53839–53858, 2019.
- [27] Grafana. Accessed September 28, 2020. <https://grafana.com/>, 2020.
- [28] Yu Liu, Zhibo Pang, Magnus Karlsson, and Shaofang Gong. Anomaly detection based on machine learning in iot-based vertical plant wall for indoor climate control. *Building and Environment*, 183:107212, 2020.