KFIML: Kubernetes-Based Fog Computing IoT Platform for Online Machine Learning

Ziyu Wan, Zheng Zhang, Rui Yin, *Senior Member, IEEE*, and Guanding Yu, *Senior Member, IEEE*

***Abstract*—The massive onsite data produced by the Internet of Things (IoT) can bring valuable information and immense potentials, thus empowering a new wave of emerging applications. However, with the rapid increase of onsite IoT data streams, it has become extremely challenging to develop a scalable computing platform and provide a comprehensive workflow for processing IoT data streams with lower latency and more intelligence. To this end, we present a Kubernetes-based scalable fog computing platform (KFIML), integrating big data streaming processing with machine learning (ML)-based applications. We also provide a comprehensive IoT data processing workflow, including data access and transfer, big data processing, online ML, long-term storage, and monitoring. The platform is feasibly validated on a clustered testbed, which comprises a master node, IoT broker servers, worker nodes, and a local database server. By leveraging the lightweight orchestration system, namely Kubernetes, we can readily scale and manage containerized software frameworks on our testbed. The big data processing layer utilizes the advanced data flow frameworks such as Apache Flink, to support both streaming processing and statistical analysis with low latency. In addition, the specified long short-term memory (LSTM)-based ML pipelines are employed on the online ML layer, to enable the real-time predictive analysis of IoT data streams. The experiments on a real-world smart grid use case demonstrate that the container-based KFIML platform can be well-scaled with Kubernetes to efficiently perform big data processing increased onsite IoT data streams with lower latency and conduct ML-based applications.**

***Index Terms*—Big data processing, fog computing, Internet of Things (IoT), Kubernetes, online machine learning (ML), predictive analysis.**

I. Introduction

THE Internet of Everything (IoE) is an emerging concept that combines data, things, processes, and people to promote the Internet of Things (IoT) with smarter data-driven applications [1] (e.g., smart grid and smart healthcare). IoE applications generate a large quantity of real-time and high-velocity IoT data streams from distributed devices [2], thus posing enormous challenges to centralized cloud computing platforms with long-distance data transmission [3] in terms of data transmission, stream processing, security, and storage [4]. To address these problems, we investigate a promising paradigm namely fog computing [5], [6] to augment cloud-based computing systems by bringing computing resources closer to the users [4] and perform data processing at the network edge. The fog computing-enabled platform has been proven to enjoy superiority in executing latency-sensitive applications at the distributed nodes [7], [8] and improving security [9]. In addition, when integrated with distributed big data frameworks, it can facilitate streaming processing, data mining, and statistical analysis of real-time IoT data streams.

The recent state-of-the-art big data frameworks, such as *Apache Flink* (Flink) and *Apache Spark* (Spark), have been developed to process real-time data streams [10] with a specified parallel programming paradigm [11]. Such parallel computing resources are available in the field of fog computing. With the help of big data processing, various types of IoT data (i.e., high-velocity power data originating from smart grids) are regarded as unbounded data streams, which can be uniformly delivered to distributed fog nodes for parallel processing to improve computing efficiency. In addition, the big data framework, especially Flink, has evolved into a unified stream/batch computing platform [12]. They are also capable of handling both onsite IoT data streams and offline historical records to promote responses, thereby empowering real-time streaming computation and batch statistical analyses.

Although big data processing with fog computing provides effective solutions to handle onsite IoT data streams, it lacks capacities to perform innovative analyses provided by machine learning (ML) techniques [13]. Therefore, the fog computing platform is expected to integrate big data processing and ML-based applications so as to fully utilize onsite IoT data streams. However, it is challenging for current fog computing platforms to combine these two aspects because the IoT data processing workflow designed for fog computing platforms may not be complete. Therefore, there is an urgent need to develop a relative comprehensive processing workflow for existing fog platforms to explore innovative applications.

A growing trend has been observed in the adoption of ML to provide efficient resource management, enhanced security, and reduced latency for various fog computing applications [14]. For instance, the use of ML models in the electricity domain has been proven to appreciably improve the accuracy and robustness of predicting energy consumption [15]. However, the current ML libraries such as Spark MLlib provided by big data frameworks have many drawbacks. For example, they cannot be easily utilized to design specified ML

Manuscript received 27 December 2021; revised 9 March 2022; accepted 10 April 2022. Date of publication 18 April 2022; date of current version 23 September 2022. *(Corresponding author: Guanding Yu.)*

Ziyu Wan, Zheng Zhang, and Guanding Yu are with the College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou 310027, China (e-mail: ziyuwan@zju.edu.cn; zheng_zhang@zju.edu.cn; yuguanding@zju.edu.cn).

Rui Yin is with the School of Information and Electrical Engineering, Zhejiang University City College, Hangzhou 310015, China (e-mail: yinrui@zucc.edu.cn).

Digital Object Identifier 10.1109/JIOT.2022.3168085

procedures [16], and they do not support multivariate regression models [17]. Moreover, the traditional ML offline batch training manner fails to meet the requirements of predictive analyses for onsite IoT data because of the long update intervals. Therefore, by leveraging widely adopted ML frameworks, such as Tensorflow, we develop a specified online ML pipeline and apply the newest model to analyze onsite data streams.

With more and more onsite IoT data streams, it is challenging for current fog computing platforms to address this issue with traditional scaling manners. Recently, container-based deployment, which is regarded as one of the most efficient and lightweight manners, has been employed to rapidly create, scale, and manage massive IoT applications on the fog platform [18], [19]. Though they are isolated application processes, containers share the underlying host operating system [20], and typically have a stronger scaling capability than virtual machines (VMs). Kubernetes [21] is a distributed container orchestration engine for automatically scaling applications and managing clustered computing resources. By leveraging Kubernetes, containerized frameworks running on distributed fog servers can communicate with others and share cluster computing resources [20]. Moreover, the master-worker architecture [22] of Kubernetes allows users to assign specific roles to heterogeneous fog servers. Current advanced big data frameworks provide various manners to automatically deploy the clustered computing environment on top of Kubernetes.

In this article, we propose a Kubernetes-based five-tier fog computing architecture, namely, KFIML, to efficiently address the challenge from processing an increasingly large body of onsite IoT data streams. It also integrates big data processing and ML-based applications in order to explore innovative analyses of real-time IoT data based on a comprehensive IoT data processing workflow. Furthermore, we implement the platform on a distributed and heterogeneous cluster to validate its feasibility. The platform is equipped with clustered and container-based software frameworks. With the orchestration of Kubernetes, it is more convenient to scale the software components on KFIML and more efficient to perform the whole IoT data processing workflow.

In our work, a real-world smart grid use case is adapted to evaluate the feasibility of the overall workflow in five-tier KFIML. We evaluate the system behavior when it scales to address the increased workload of processing larger-scale onsite phasor measurement unit (PMU) data streams. At the big data processing layer, we conduct big data processing tasks, including streaming power computation and statistical analyses. To further conduct predictive analyses, a specified ML pipeline is built on the online ML layer, which comprises online training, model validation, and online prediction processes. The evaluation results show that the container-based KFIML platform performs IoT stream processing with relatively lower latency and better scalability.

In summary, the main contributions of this article are as follows.

1) We propose a five-tier fog computing platform architecture to integrate big data stream processing and ML-based predictive analyses of high-velocity onsite IoT data streams. Compared with those existing works that conduct either ML-based application or stream processing, our platform presents a relative comprehensive workflow, including IoT data access and transfer, stream processing, online ML, long-term storage, and real-time monitoring.
2) We deploy a heterogeneous fog computing testbed to implement the multitier KFIML, and leverage Kubernetes as the orchestration tool to guarantee the scalability of containerized frameworks.
3) A real-world smart grid use case is adapted for the KFIML platform to process high-velocity IoT data streams. We realize streaming power computation and statistical analyses on the big data processing layer for onsite multisource PMU data streams. A long short-term memory (LSTM)-based online ML pipeline is employed on the online ML layer to forecast multizone power parameters. The experimental results verify its feasibility in conducting both big data processing and online ML applications on fog computing testbed.
4) We evaluate the latency of streaming processing with increased quantity of real-time PMU data sources. The experimental results show that our structured KFIML outperforms Spark Streaming in terms of processing latency with different parallelisms, and achieves a better scalability performance than non-Kubernetes deployment manner.

The remainder of this article is organized as follows. In Section II, the related works are introduced. Section III outlines the multitier architecture of the proposed KFIML platform. In Section IV, we implement the container-based KFIML platform on a clustered fog testbed. Section V presents a smart grid use case to functionally validate the platform. Section VI provides the evaluation details and experimental results. Finally, we conclude this article and discuss future work in Section VII.

II. Related Work

In this section, we investigate existing realistic fog computing platforms applied in IoT scenarios and divide recent work into two categories: 1) big data processing and 2) ML-based analysis. However, a comprehensive workflow of processing real-time IoT data streams has not been developed in the existing fog platforms, such as [23]–[25]. In addition, several works have considered scalability and latency performances when handling the increased workload of large-scale onsite IoT data streams, such as [2], [26], and [27].

A. Big Data Processing on Fog Computing Platform

Nguyen *et al.* [23] proposed a two-tier fog computing architecture to achieve streaming processing in IoT applications. On the fog layer, they implement two-tier Kafka and Spark clusters named A-Fog and S-Fog on Raspberry-pi boards. They leverage Spark to conduct real-time streaming processing in a smart parking scenario and calculate the onsite park-slot status every minute. Their work demonstrates that the fog platform itself can conduct both streaming processing and large-scale

data analyses on real-time IoT data. However, their platform fails to notice the workflow of parsing IoT data by gateways according to IoT protocols. In addition, the real-time processing latency has not yet been gauged.

de Mello Alencar *et al.* [2] introduced a three-layer fog computing IoT platform, which includes a device layer, a network edge layer, and a cloud layer to support real-time IoT data stream processing. Their platform is designed primarily to reduce the amount of data transmitted to the cloud layer by using traditional signal processing methods, such as wavelet transform and concept drift detection. The edge layer is composed of distributed Raspberry Pi boards which implement Kafka and MQTT servers for stream processing and IoT data parsing, respectively. A real-world smart building scenario is applied to their platform with 15 sensors collecting and sending data to the network edge layer every 9 s. However, the scalability of their platform has not been evaluated.

The work in [24] presents a multitier fog computing platform architecture to analyze large-scale data originating from smart city scenarios. Their platform is implemented on "A-Fogs" (a cluster of eight RPi boards) and "D-Fogs" (a cluster of four powerful servers) with Spark engine installed in VMs. They utilize two algorithms, including support vector machine and logistic regression to conduct analyses on two different data sets. However, the authors only evaluate the scalability of their VM-based platform with the increasing number of fog nodes and larger stationery data sets generated by Spark.

B. Machine Learning for Onsite Data Processing

Akbar *et al.* [13] proposed and implement a generic architecture based on open source components to conduct predictive analyses and detect complex events from IoT applications. In their platform, Node-RED is utilized to receive and parse real-time MQTT messages and the data streams are then transferred to Kafka brokers. They adopt a real traffic features measurement application, designed a prediction algorithm for near real-time IoT data, and implement it with Python Scikit-learn and Spark MLlib. However, they have not evaluated the scaling performance when their platform is applied to analyze the massive influx of IoT data.

Omran *et al.* [28] proposed an ML pipeline for predicting breast cancer disease in real time. Based on Spark and Kafka, the system consists of two parts, including offline model training and online prediction. The testbed is a VM-based cluster, with 7 CPU cores, 20-GB RAM, and 100-GB disk storage. On this basis, the online prediction procedure loads the offline trained model on Spark Streaming. Then, it makes predictions based on real tweet streams obtained from Kafka brokers. The experimental results demonstrate the feasibility of online predictive analyses of onsite data streams. However, the authors have not evaluated the scalability of online ML pipelines on their clustered testbed.

A hierarchically distributed fog computing architecture, which supports ML-based real-time anomaly detection for smart meter data, is proposed in [27]. The fog platform consists of the "core layer" and the "edge layer," which trains prediction models and conducts real-time anomaly detection, respectively. The testbed includes two dual-core Intel i5 computers as the two layers. The experimental results indicate that the running time of the algorithm ensures the feasibility of real-time anomaly detection. However, the authors have not evaluated the scalability of their distributed fog computing architecture by using multisource household data sets.

III. MUTLILAYER KFIML PLATFORM ARCHITECTURE

In this section, we provide the overall architecture of the proposed multilayer KFIML platform. The motivation of this work is to establish a comprehensive fog computing platform capable of handling high-velocity onsite IoT data streams. In this context, we integrate big data processing and online ML in order to realize analytical intelligence of fog computing-assisted IoT platforms based on limited computing resources. However, it is difficult to control IoT data flows on such an intelligent fog platform because the current big data and ML frameworks are not able to directly extract the information from IoT data streams. To this end, the architecture can be divided into five tiers (as depicted in Fig. 1), including an IoT data access layer, a data transfer layer, a big data processing layer, an online ML layer, and a data storage layer.

The IoT data access layer aims to acquire and parse original IoT data messages generated by heterogeneous IoT terminal devices (e.g., IoT gateways), and then to bridge required data payloads to the next layer. IoT brokers provide rule engines that can be developed to extract useful data from IoT messages encoded by various protocols such as MQTT. After that, the data transfer layer is responsible for data bridging between the IoT data access layer and the big data processing layer through high-throughput distributed messaging system such as Apache Kafka[1] (Kafka).

On the big data processing layer, we employ the advanced distributed data flow frameworks, such as Apache Flink,[2] to support the stream processing of high-velocity IoT data streams. Flink fetches data messages from Kafka brokers, performs streaming data transformation, conducts window aggregation, and outputs results to user-defined destinations (e.g., time-series database). Furthermore, the online ML layer utilizes the most widely adopted ML framework, Tensorflow, to develop a specific ML procedure for the online prediction of processed data from the big data processing layer. The online prediction process continuously loads the newest trained model, and then conducts predictive analyses for onsite data streams. Last but not least, the results of the big data processing layer and the online ML layer are streamed to the data storage layer immediately for long-term storage. Meanwhile, the real-time monitoring dashboard continuously queries the latest historical analytical results and displays them on time-series graphs.

To the best of our knowledge, there are very few distributed fog computing platforms that integrate both big data stream processing applications and online ML pipelines to explore the innovative analysis of onsite IoT data streams. In this article, we provide a more comprehensive workflow for real-time IoT

[1]https://kafka.apache.org/

[2]https://flink.apache.org/

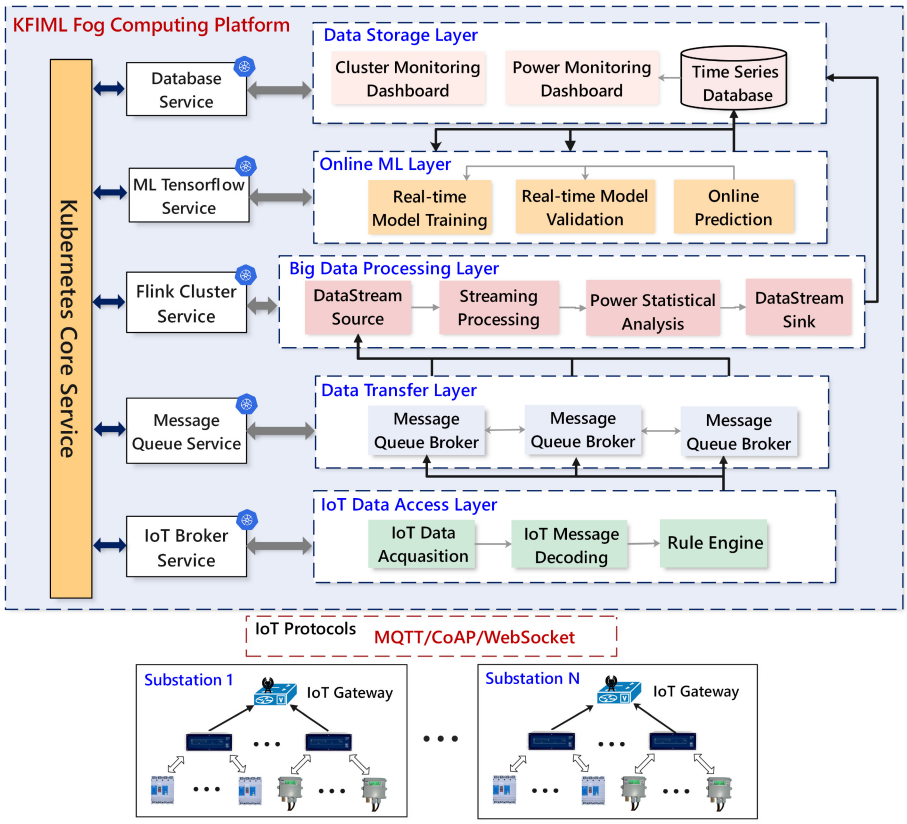


Fig. 1. Proposed multilayer architecture of fog computing platform.

TABLE I
COMPARISON OF STATE-OF-THE-ART FOG COMPUTING IOT PLATFORMS

Fog Platform	Dataset types	Multiple data sources	Cluster management[1]	Stream processing	ML-based analysis	IoT data access[2]	Scalability	Data sending time interval	Long-term storage
Testbed in [23]	Real-time simulated	✓		Spark +Kafka			✓	about 1 minute (medium)	✓
Testbed in [2]	Real-time simulated	✓		Kafka		Reactive MQTT broker		about 9 seconds (medium)	✓
Testbed in [24]	Stationery datasets		✓	Spark	SVM+LR		✓		
Testbed in [13]	Real-time simulated	✓		Kafka +Esper	Predictive analysis	Node-RED		about 5 minute (low)	
KFIML testbed	Real-time simulated	✓	✓	Flink+Kafka	Predictive analysis	EMQ MQTT broker	✓	about 50 ms (high)	✓

[1] The software frameworks running at distributed fog servers are managed by orchestration systems such as Kubernetes.
[2] The fog computing platform provides the access to concurrently collect and transfer real-time IoT data messages.

data processing tasks with a multitier architecture. Meanwhile, the orchestration and management of container-based software frameworks by Kubernetes on the Fog testbed guarantees the scalability of KFIML when handling increased data processing workload. The comparisons between KFIML and the state-of-the-art fog computing platforms are shown in Table I.

IV. TESTBED IMPLEMENTATION

The physical architecture of the KFIML testbed is illustrated in Fig. 2 and its experimental build-up is shown in Fig. 3. It consists of four kinds of fog servers, including multiple IoT broker nodes, three worker nodes, a local database server, and a master node. These fog servers are connected to a switch via high-speed Ethernet cables (up to 1 Gb/s) to reduce data communication latency when working in parallel. The Raspberry-pi single-board computers equipped with light-weight MQTT brokers and Kafka brokers serve as IoT broker servers. Due to limited computing resources, they are setup closest to IoT gateways, and support collection, parsing and transfer of multisource IoT messages.

We apply the open-source and clustered IoT message broker framework, EMQ X broker[3] to our testbed. It provides SQL-based built-in rule engines, which can parse data messages according to IoT protocols (e.g., MQTT), and then stream the required data to databases or MQ brokers such as Kafka [29]. Furthermore, clusters of EMQ X and Kafka brokers are both deployed with Kubernetes *StatefulSet* resources, to guarantee automated scaling and persistent storage of these brokers [30].

The detailed hardware configurations and software specifications of fog servers are listed in Table II. Since our

[3]https://www.emqx.io/products/broker

TABLE II
HARDWARE CONFIGURATIONS AND SOFTWARE SPECIFICATIONS OF KFIML TESTBED

Features	IoT broker Servers	Worker Nodes	Master Node	Database Server
Node Type	Three Raspberry 4B computers	Three x86 computers	One x86 computer	One x86 computer
CPU Specifications	Cores: 4 Architecture: ARM64 V8 Model: Cortex-A72 Soc: 64-bit, 1.5GHz Thread 4 BogoMIPS: 108 Max MHZ: 1500	Cores: 6 Model: Intel Core i7-8750H Threads: 12 OS Type: 64 CPU MHz: 800 Max MHz: 4100 BogoMIPS: 4399.9	Cores: 6 Model: Intel Core i7-8750H Threads: 12 OS Type: 64 CPU MHz: 800 Max MHz: 4100 BogoMIPS: 4399.9	Cores: 6 Model: Intel Core i5-8500 Thread(s): 6 OS Type: 64 CPU MHz: 3961.147 Max MHz: 4100 BogoMIPS: 6000
Resources	RAM: 4GB Storage: 32GB	RAM: 7.62GB Storage: 250GB	RAM: 7.62GB Storage: 128GB	RAM: 7.62GB Storage: 180GB
Operating System	Ubuntu 20.04	Ubuntu 20.04.2 LTS	Ubuntu 20.04.2 LTS	Ubuntu 20.04.2 LTS
Flink Specifications		version: 1.12.1-scala_2.12 role: Task Manager memory process size: 1728m rpc port: 6122	version: 1.12.1-scala_2.12 role: Job Manager memory process size: 1600m rpc port: 6123	
Flink Development		setup: Flink on k8s cluster mode: session cluster	setup: Flink on k8s cluster mode: session cluster	setup: Flink on k8s cluster mode: session cluster
Kubernetes Specifications	version: 1.20.6+k3s1 Goversion: 1.15.0	version: 1.20.6+k3s1 Goversion: 1.15.0	version: 1.20.6+k3s1 Goversion: 1.15.0	version: 1.20.6+k3s1 Goversion: 1.15.0
Kubernetes Roles	iotbroker	worker	master, control-plane	worker, database server
Software Components (kind, default No.)	+Kafka 2.0.0 (StatefulSet, 3) +Zookeeper 2.0.0 (StatefulSet, 3) +MQTT Broker (StatefulSet, 3)	+Flink Task Manager (Deployment, 3) +Tensorflow-LSTM Predictor (Deployment, 6)	+Flink Job Manager (Deployment, 1)	+Influxdb1.7 (Deployment, 1) +Grafana (Deployment, 1)

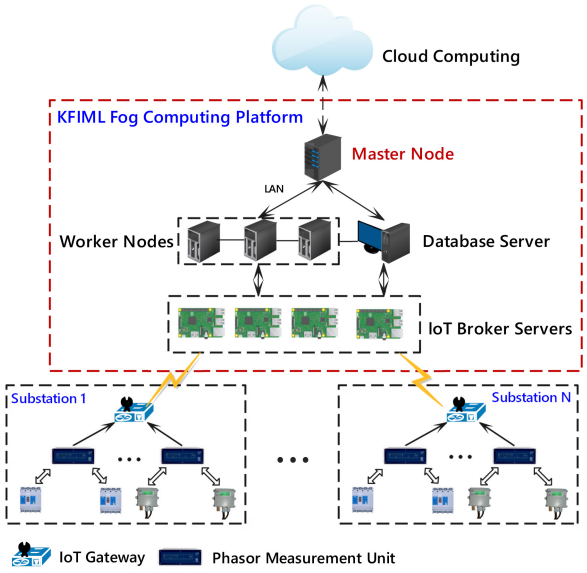


Fig. 2. Physical architecture of the KFIML fog computing platform.

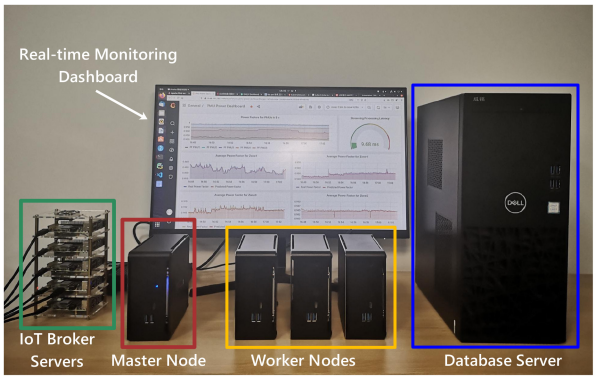


Fig. 3. Testbed build up.

testbed has heterogeneous computing resources, we define the *nodeSelector* field in the yaml files to specify which node to deploy data processing tasks on different layers according to node labels. The worker nodes cluster, composed of three ×86 computers, is built up to execute big data processing tasks as well as online ML applications in a distributed environment. Compared with light-weight Raspberry-pi computers, they have more computing resources (six CPU cores and 12 threads), a higher computation speed (BogoMIPS up to 4400), and a larger available memory (7.62-GB RAM and 250-GB disk storage). The operating systems used by these servers are all Ubuntu 20.04, thereby ensuring that computing environments on the distributed nodes are consistent.

On the testbed, the Flink standalone cluster is deployed with the official recommended *session mode*, and includes a job manager (JM) and multiple task managers (TMs). They are executed as native Kubernetes *deployments* [31] and scheduled to the master node and the work nodes, respectively. In addition, our testbed establishes a comprehensive IoT data processing workflow, so that developers only need to focus on developing specific stream processing applications with predefined data sources (e.g., Kafka brokers) and data destinations (e.g., Influxdb). Flink enables users to develop streaming processing, batch processing, and event driven applications with provided functional Java, Scala, and Python APIs (e.g., DataStream API). To run these applications, developers need to submit complied packages such as Jar packages to Flink JM and set the running parallelism according to

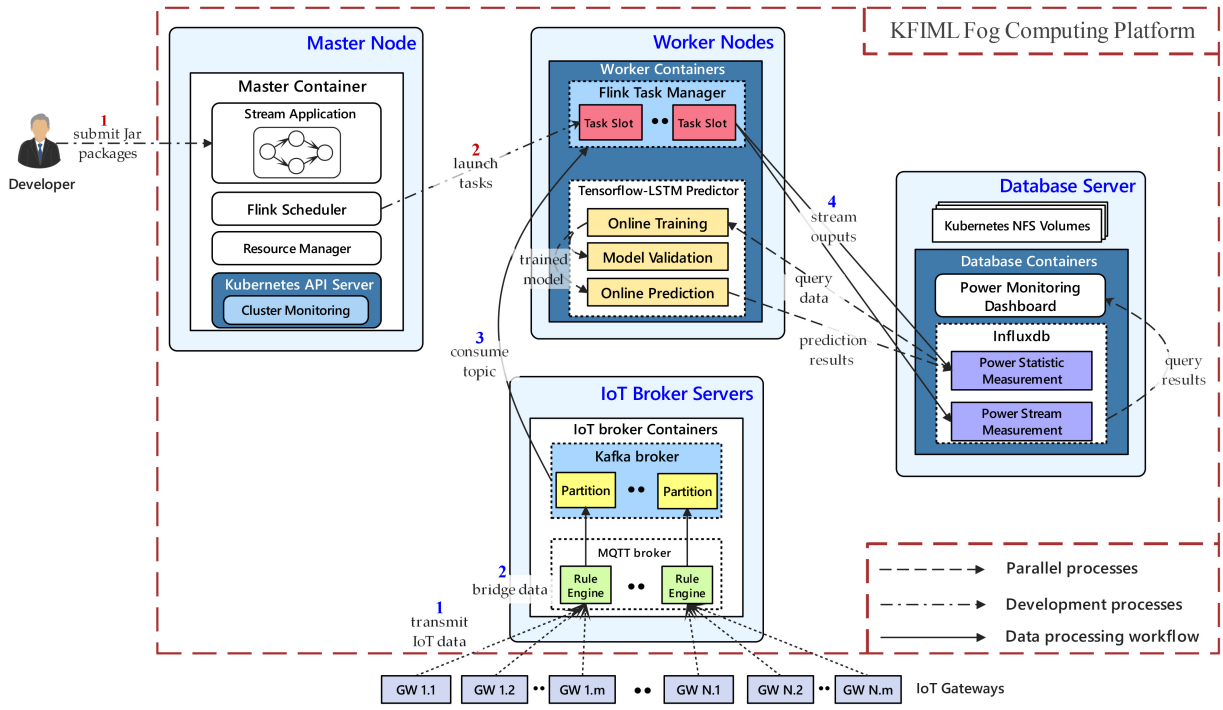


Fig. 4. Overview of software containers and data flows in the Kubernetes cluster.

available computing resources. After that, these applications are converted to dataflow graphs at the master node, and then are scheduled to multiple *task slots* (parallel computing resources) of Flink TMs at the worker nodes.

The workflow and frameworks of the overall architecture of the KFIML testbed are depicted in Fig. 4. Kubernetes is a really complicated orchestration system which continuously manages the life cycles of container-based frameworks in KFIML. The IoT brokers aim to parse the onsite IoT protocol messages transmitted from IoT gateways, obtain the data payloads, and then stream required data fields to Kafka brokers on the data transfer layer. For big data processing tasks, we need to define data sources (e.g., Kafka), sinks (e.g., Influxdb) and data transformation logics first, and then compile and submit Flink jar packages to Flink JM. The *task slots* resources in distributed Flink TMs are assigned to automatically perform the stream processing tasks on IoT data streams. In addition, a parallel ML pipeline is developed for the online prediction of onsite data with LSTM-based ML algorithms. This pipeline is running in an integrated container, named as "tensorflow-LSTM predictor," including online training, model validation, and online prediction. The online training process primarily queries the historical time-series data records from Influxdb,[4] divides the obtained data into a training set and a validation set, and then trains the LSTM model with predefined parameters, such as batch size and learning rate. The online prediction process loads the newest LSTM model, applies it to conduct online inference on real-time data, and finally outputs the predicted values to Influxdb on the database server for long-term storage. Grafana[5] provides the front-end where real-time results from big data stream processing and online prediction are visualized. In addition, on the database server, we establish a cluster network file system (NFS) to enable the persistent storage of Kafka topic partitions, Tensorflow ML models, EMQ X rule engines, and Influxdb time-series databases.

V. Smart Grid Use Case

To evaluate the feasibility of the proposed platform in processing high-velocity IoT data streams, we consider a real-world smart grid application where PMUs are deployed at grid substations to continuously sense PMU data (phasor parameters) across the power grid at a sampling rate of 30–120 samples/s [32]. Then, all PMU data are streamed to control centers [33]. The raw PMU data comprises three-phase voltages, three-phase currents, and phase angles. Moreover, the increasing amounts of real-time and high-velocity PMU data contribute significantly to increasing latency due to high computational complexity [32]. Therefore, this latency-sensitive application can be deployed on the fog platform because of low computation costs and short network transmission delays [34]. In actual scenarios, the power state of each substation is essential for detecting abnormal events and conditions. Therefore, it is beneficial to process the raw PMU data to obtain the real-time power parameters (e.g., active power, reactive power, power factor, and apparent power).

The motivation for processing high-velocity IoT data streams stems from an open-access campus PMU data set given by the EPFL smart grid project [35], which is carried out at EPFL University in Switzerland. On the EPFL campus, there is a smart grid system where six buildings are monitored via a state estimation mechanism powered by six PMUs [35].

[4]https://www.influxdata.com/products/influxdb/
[5]https://grafana.com/grafana/dashboards

Each PMU continuously senses the above-mentioned electrical parameters on the substation at an interval of approximately 20 milliseconds (ms). The data set contains historical records for six PMUs labeled as PMU0-PMU5 in the selected months from 2014 to 2020. In this work, we utilize the data records sensed by six PMUs from 12:00 P.M. to 1:00 A.M. on March 23, 2019. Each data set is composed of 65 536 raw PMU data records. Specifically, we regard each PMU as a grid substation zone, aiming to scale the simulators based on zone identifier (ID) in subsequent experiments.

A. Simulators for PMUs and IoT Gateways

Instead of utilizing real PMUs and IoT gateways in the experiment, we apply a light-weight thread, *goroutine*, in Golang language to develop simulators running in parallel. These simulators aim to imitate the process that PMUs continuously collect real-time phasor parameters on power lines. Then, the IoT gateway sends PMU data in the grid substation to our KFIML platform through the MQTT protocol. At each time interval, simulators first extract data fields from corresponding EPFL PMU data sets and then create JSON messages with the following fields: 1) timestamp when PMU data is generated; 2) PMU ID; 3) zone ID; and 4) PMU data payload. In addition, the IDs of PMUs in the first zone are marked from PMU100 to PMU199, and PMU200-PMU299 represent all possible PMU IDs in the second zone. Then, we utilize the Eclipse Paho MQTT client library[6] to establish multiple MQTT clients with the following parameters: 1) MQTT topic; 2) client ID; 3) IoT brokers addresses; and 4) Quality of Service (QoS) level. The number of MQTT clients is equal to that of PMU data sources. Each client imitates a data connection established by IoT gateways for streaming one set of PMU data to IoT brokers. After a set of PMU data is generated, the client encapsulates it into an MQTT message, and then publishes the message to IoT brokers cluster through wireless or wired communication methods.

The MQTT protocol defines three levels of QoS, including 0, 1, and 2, which represent *at most once*, *at least once*, and *exactly once*, respectively [36]. In particular, the higher the level of QoS is, the less the probability of data loss will be, but the latency of MQTT message transmission will be larger. Given that the real-world data collection and transmission delays can not be ignored, each of these simulators is configured in a light-weight *goroutine*, collects PMU data in a time interval of 20 ms, and sends MQTT messages to the IoT brokers via low-latency Ethernet every 50 ms.

B. Rule Engine and Kafka Broker

As mentioned in Section IV, the EMQ X broker provides users with an SQL-based rule engine which can be developed manually to bridge the required data fields to the specific topics of Kafka brokers with the following parameters: 1) *Kafka cluster bootstrap servers*; 2) *topic name*; 3) *max batch processing payload size*; and 4) *partitioning strategy*. In this experiment, a Kafka topic "PMUdata" is established to receive the data streams transmitted by rule engines. By leveraging a universal Kafka connector [37] defined in Java, the data streams are immediately obtained by Flink TM programs running at the worker nodes.

In this work, each PMU is imitated by a real-time simulator, which is composed of a PMU data generator and an MQTT client with a specified MQTT topic. For instance, the topic "ZONE1/PMU/100" refers to the PMU with ID 100 at Substation Zone 1. Furthermore, each message matches one of six rule engines according to its MQTT topic, which ranges from "ZONE1/PMU/+" to "ZONE6/PMU/+." Here, the symbol "+" acts as a topic filter, which is to extract all the data streams originating from multiple PMUs at one substation zone. The number of Kafka partitions is set to 6, which indicates that each Kafka partition is associated with an EMQ X rule engine. Then, those PMU data continuously parsed from six substation zones are transmitted to these six Kafka partitions, respectively. The multizone PMU data are stored in isolated NFS dictionaries at the database server in a short time, and then Flink TMs fetch the newest data and conduct streaming power computing applications.

C. Streaming Power Computation

The power parameters adopted in this application include active power, reactive power, apparent power, and power factor, which can be calculated based on the magnitudes of three-phase voltages V_A, V_B, and V_C, three-phase currents I_A, I_B, and I_C, and corresponding phase angles. As shown in (1), the apparent power *App* is the sum of reactive power *Rea* and active power *Act*. Power factor *PF* is the ratio of active power to apparent power, and it is a widely adopted parameter to reflect how effectively is the incoming power utilized in a power grid system

$$\begin{aligned} \text{Act} &= V_A I_A \cos\phi_A + V_B I_B \cos\phi_B + V_C I_C \cos\phi_C \\ \text{Rea} &= V_A I_A \sin\phi_A + V_B I_B \sin\phi_B + V_C I_C \sin\phi_C \\ \text{App} &= V_A I_A + V_B I_B + V_C I_C, \quad PF = \text{Act}/\text{Rea} \end{aligned} \tag{1}$$

where ϕ_A, ϕ_B, and ϕ_C denote the differences between phase angles (in degree) of three-phase voltages and corresponding currents. Algorithm 1 describes the process of streaming power computation application on real-time PMU data. As mentioned in Section V-A, the simulator adds a timestamp T_g, namely, "EventTime" on the original data message when the PMU data is generated. To measure the streaming processing latency L_p, we add another timestamp T_p, named as "ProcessTime" on each power record when the power computation is accomplished. This latency is then calculated by subtracting these two timestamps and recorded in a JSON field named as *DelayTime* in each output power record.

Meanwhile, the results of streaming power computation are immediately streamed to the *measurement* "PMUStream" of the database "PMUPower" for long-term storage. Given that the Flink-Influxdb connector is not officially provided, we utilize Flink *RichSinkFunction* and configure the Influxdb Java client[7] with the following parameters: 1) Influxdb address (IP

[6]https://github.com/eclipse/paho.mqtt.golang

[7]https://github.com/influxdata/influxdb-java

Algorithm 1 Streaming Power Computation on Real-Time PMU Data at Worker Nodes

Input: Real-time and multi-source PMU data streams.
Output: Real-time power data streams.

1: **Initialization:**
2: Initialize the Flink Stream Execution Environment with specific configurations such as checkpoint interval, parallelism and timeout interval.
3: Initialize the Kafka consumer with specific configurations: source topic, bootstrap server address and offset etc.
4: Initialize the Influxdb database connection with address and database names.
5: Define the schema for calculating power parameters including the Apparent Power *App*, Active Power *Act*, Reactive Power *Rea*, and Power Factor.
6: **On arrival of a PMU data from Kafka:**
7: Map PMU electrical parameters to power parameters, add the timestamp T_p when power computation is finished.
8: Calculate the streaming processing latency $L_p = T_p - T_g$.
9: Stream these power results including App, Rea, Act, PF and latency L_p to Influxdb for long-term storage.

address and port); 2) database and measurement names; 3) data fields (e.g., *Apparent Power* and *Latency*); and 4) tag (e.g., *ZoneID*). The average latency within 5 s is continuously queried by Grafana, and then visualized on the real-time monitoring dashboard. Instead of setting timestamps manually, Influxdb automatically adds a timestamp for each power record. In addition, we set the *ZoneID* as the *tag* to distinguish from the powers of PMUs in multiple zones.

Given that the high-velocity and real-time powers are not actually needed for grid managers because the large-scale power records cannot be observed clearly, we need to develop a statistical analysis application to provide statistical features of these powers. The next application conducted in the big data processing layer aims to continuously obtain the statistical features, including the average, maximum, and minimum values of these power parameters within a specific time interval.

D. Power Statistical Analysis

This application aims to provide grid managers with a statistical analysis tool to evaluate the real-time power state of each substation zone. Algorithm 2 presents the workflow of power statistics applications with Flink *Window* functions. We first leverage the Flink key selector, *keyby()*, to split the real-time power data streams (output of streaming power computation application) into groups according to the *ZoneIDs*.

As illustrated in Fig. 5, each two groups are then processed by two available Flink parallel *task slots* at one worker node. Then, we utilize Flink *SlidingEventTimeWindow* operator to aggregate power records arriving at a specific time interval, and calculate the statistical features of power parameters (e.g., power factor). In addition, the *sliding window* continuously slides forward with a fixed length, and statistical power results are then obtained at the end of each window.

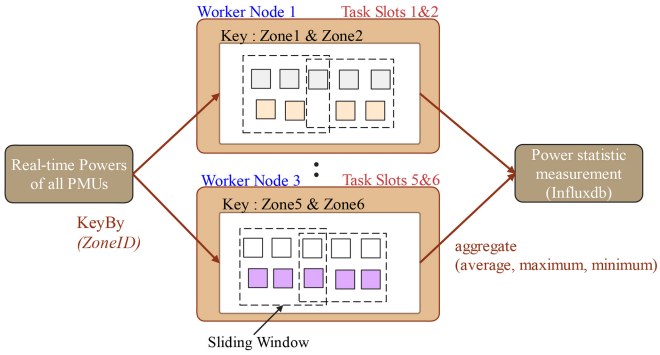


Fig. 5. Process of power statistical analysis with sliding window.

Algorithm 2 Power Statistics Application on Real-Time Power Records at Worker Nodes

Input: Real-time power data streams.
Output: Statistical features of multi-zone power data.

1: **Initialization:**
2: Define a Java class namely "WindowBean" to record the statistical power features.
3: Initialize sliding event time windows according to *ZoneID* and define window sizes and sliding sizes.
4: **On arrival of a power result:**
5: Obtain the current power R_p and update the newest maximum Max_p, minimum Min_p, count $C_p = C_p + 1$ and sum $Sum_p = Sum_p + R_p$ for each power parameter.
6: **At the end of each sliding window:**
7: Calculate the average value $Avg_p = Sum_p/C_p$ and obtain the final Max_p, and Min_p for each power parameter.
8: Collect Avg_p, Max_p, Min_p, *PMUID*, *ZoneID* and window start time in an instance of class "WindowBean".
9: Stream collected results to the measurement "PMUWindow" in Influxdb database for long-term storage.
10: **Real-time power monitoring:**
11: Grafana continuously queries the average power factors through the tag *ZoneID*, and then displays the average power factors of six zones in time series graphs.

Finally, these statistical results are streamed to the measurement "PMUWindow" of the Influxdb database "PMUPower" for long-term storage. In this experiment, the real-time average power factors of multiple substation zones are displayed on the Grafana dashboard as depicted in Fig. 6.

E. Machine Learning for Online Prediction

To forecast the upcoming power parameters of substation zones, we develop an online ML pipeline (as described in Algorithm 3), which utilizes a four-layer LSTM network. The LSTM network, which is a variant of recurrent neural networks, performs better in processing sequential data [38] such as onsite IoT data streams.

As mentioned in Section IV, the ML-based predictive analysis pipeline is composed of three aspects: 1) online training; 2) model validation; and 3) online prediction. The overall ML pipeline is mainly developed with Javascript and

Algorithm 3 Online ML for Predicting Power Factors at Worker Nodes

Input: Real-time statistical power data.

Output: Predicted power factors of all zones.

1: Initialize an LSTM prediction model including four LSTM cells and four dropout layers.
2: Define the number of units at each layer, optimizer, epochs, batch size, learning rates, prediction sequence length n and the number of predicted values s.
3: **Online Training Process:**
4: Obtain current time t, and query the historical 10-minute average power factors $S(t)$ from Influxdb.
5: Randomly select 80% of $S(t)$ as the training set $Tr(t)$ and 20% as the validation set $Va(t)$, then transform $Tr(t)$ by *MinMaxScalar*.
6: **for** $i = n \rightarrow Tr(t).length$ **do**
7: Push $Tr(t)[i - n \rightarrow i]$ to model input $X_{train}(t)$ and $Tr(t)[i]$ to target $y(t)$.
8: **end for**
9: Train the model with $X_{train}(t)$, $y(t)$, then save the model weights at the Kubernetes volumes.
10: **Online Prediction Process:**
11: Set prediction sequence $X_{predict}(t)$ by querying the latest n power factors of specific zone.
12: **for** $i = 1 \rightarrow s$ **do**
13: Load the newest model weights on $X_{predict}(t)$ to predict the next power factor, and stream it to *PredictedPower* measurement in Influxdb.
14: Push the predicted power factor to $X_{predict}(t)$ and drop the earliest value.
15: **end for**

APIs provided by open-source hardware-accelerated library Tensorflow.js.[8] These three processes are running concurrently in a container, which can be easily scaled and managed by the Kubernetes server at the master node.

The training and validation sets are obtained by querying the latest 10-min historical average power factors from Influxdb, where the newest outputs of power statistical analysis applications are stored. Instead of developing a comprehensive prediction model for all zones, we create six isolated containers, each of which is composed of these three ML processes. These containers are named as "Tensorflow-LSTM-Zone1" in sequence, and they are uniformly assigned to three worker nodes. For example, online prediction containers for zones 1 and 2 are assigned to the worker node 1. In addition, the model weights and structural information are stored and updated frequently in the isolated NFS volumes on the database server. The training process updates the model weights after completing all batches, and the online prediction process then loads the latest trained model to predict the future power factors. Moreover, the LSTM model is composed of four LSTM layers and four dropout layers, and each LSTM layer includes fifty LSTM cells.

[8]https://github.com/tensorflow/tfjs

TABLE III
MINIMUM MAPE (%) FOR ZONE DATA SOURCES

Data Source	Window Size	Slide Size	Learning Rate	MAPE(%)
Zone 1	2s	1s	0.05	0.18
Zone 2	5s	2.5s	0.001	0.21
Zone 3	2s	1s	0.005	0.028
Zone 4	2s	1s	0.01	0.066
Zone 5	5s	2.5s	0.001	0.16
Zone 6	2s	1s	0.01	0.091

Fig. 7 shows the prediction effect and convergence performance of the loss function, i.e., mean square error (MSE) with different window sizes, slide sizes of stream processing, and learning rates of the ML training process for all zones. The prediction effect is quantitatively represented by the mean absolute percentage error (MAPE) between the actual power factors and the predicted values of the validation results. MAPE is a statistical metric for evaluating the performance of a predictive model [13], and can be derived as

$$\text{MAPE}(\%) = 1/n \sum_{t=1}^{n} \left| \left(\frac{A_t - P_t}{A_t} \right) \right| \times 100 \tag{2}$$

where A_t and P_t denote the actual and predicted values of parameters when the time is t, respectively. The total length of the prediction sequence is n, which also represents the length of the validation set. It can be observed from Fig. 7(a)–(c) that a smaller window size and a smaller slide size can achieve better MSE performance. The reason is that a shorter length of sliding window results in a higher frequency of Flink statistical analysis and a larger amount of power data in the training set. Thus, the convergence performance becomes better. However, a larger training set leads to more time spent in obtaining the newest LSTM-based prediction model, and the smaller time interval between the first predicted value and the latest actual value.

From Fig. 7(d)–(f), we can observe that each zone has a combination of these parameters to achieve the best prediction effect. It is because the characteristics of onsite statistical power factors depend on the real power conditions of substation zones. In addition, Table III shows the corresponding window size, slide size, and learning rate when the minimum MAPE is achieved. The result indicates that grid managers need to carefully select the appropriate combinations for each substation zone. Also, more attention should be paid to configure both the streaming power computation and the ML training process to strike an optimal balance between the desired prediction time interval (e.g., from now to future 20 s) and the prediction accuracy.

VI. PERFORMANCE EVALUATION

As depicted in the previous section, we leverage our proposed KFIML platform to integrate big data processing and ML-based predictive analysis for a real-world smart grid application and test its feasibility. In this section, we further test the latency performance and the scalability of KFIML through CPU usage, streaming processing latency, and scaling time. We examine the running performance of multiple

Fig. 6. Dashboard of real and predicted power factors as well as streaming processing latency.

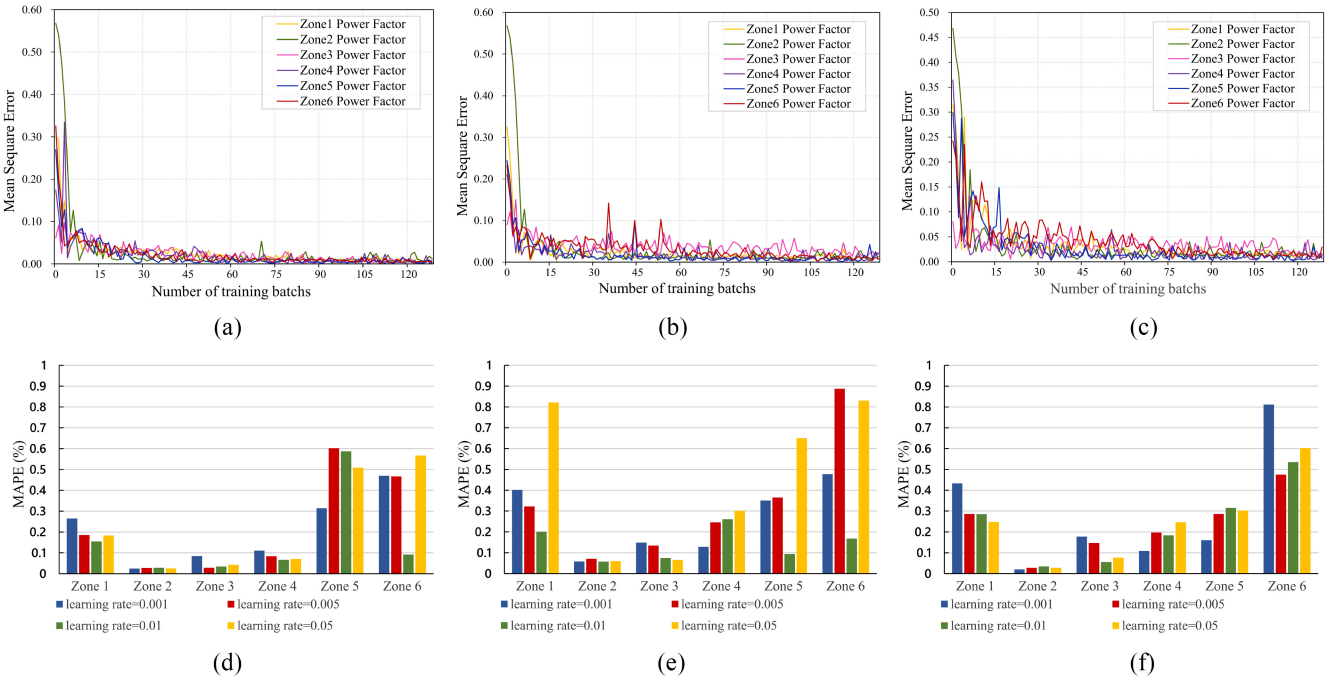


Fig. 7. Prediction effect and training performance with different window sizes, slide sizes, and learning rates. (a) Convergence performance (window size = 2 s and slide size = 1 s). (b) Convergence performance (window size = 2 s and slide size = 2 s). (c) Convergence performance (window size = 5 s and slide size = 2.5 s). (d) Prediction MAPE (window size = 2 s and slide size = 1 s). (e) Prediction MAPE (window size = 2 s and slide size = 2 s). (f) Prediction MAPE (window size = 5 s and slide size = 2.5 s).

layers when performing corresponding data processing tasks. The cluster monitoring dashboard, which is developed by Grafana and Prometheus,[9] aims to measure the real-time CPU and memory utilization of distributed Fog nodes. Fig. 8 shows the average CPU usages of four-type Fog servers when these multilayer processing tasks are gradually executed. For instance, the "ready" state represents the condition when all software frameworks are ready for onsite IoT data streams

[9]https://prometheus.io/

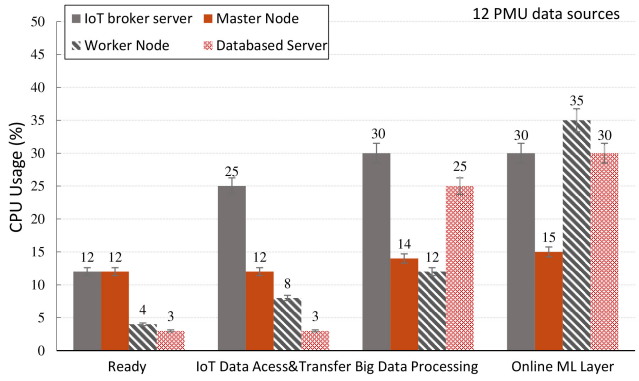


Fig. 8. CPU usages of multilayer applications running on distributed fog nodes.

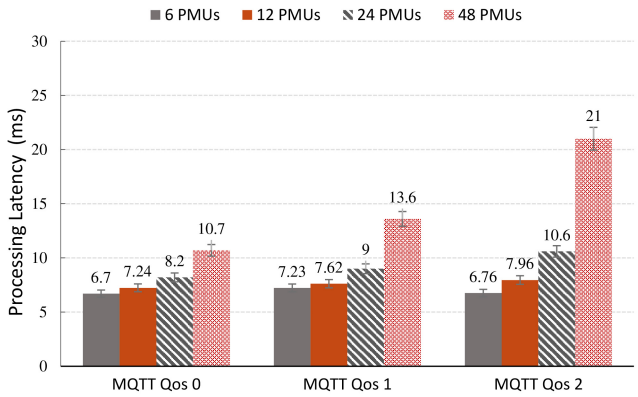


Fig. 9. Latency performance with three MQTT QoS levels and different numbers of PMU data sources.

to access the platform and the "IoT data access and transfer" stage represents that EMQX brokers parse and bridge the multizone PMU data to the Kafka cluster according to multiple rule engines.

The database server utilizes about 25% of CPU resources to store power statistics results when the streaming power computation and statistical analysis applications are both running. Meanwhile, the worker nodes consume 12% CPU to conduct big data processing tasks. As mentioned previously, two containerized online predictive analysis pipelines are running independently at one worker node. We observe that the average CPU usages of worker nodes grow from 12% to 35%. When all applications are running on the testbed, the cluster CPU usage does not exceed 35%. Therefore, the KFIML testbed has enough computing resources to conduct a comprehensive IoT data processing workflow.

We also compare the average processing latency of streaming power computation with three different MQTT QoS levels. As mentioned in Section V-III, latency is defined as the time difference between when simulators generate the phasor parameters (e.g., three-phase voltages) and when the streaming power computation based on the real-time PMU data is finished. It is at least composed of three parts: 1) data transmission from gateways to IoT brokers; 2) rule engines which parse and bridge PMU data to Kafka brokers; and 3) streaming power computation by Flink.

As shown in Fig. 9, the increasing workload in processing more PMU data sources results in relatively increased latency. Furthermore, we observe that MQTT QoS 1 and QoS 2 cause higher processing latency because they have more response mechanisms to ensure that every message arrives at the IoT broker *at least once* and *exactly once*, respectively, thereby resulting in a higher data transmission latency. When the number of PMU data sources reaches eight times of the original EPFL smart grid data sets, KFIML can still provide high performance with a mean processing latency around 20-ms under the highest level of MQTT QoS. This latency is equal to the time interval of PMU data generated in a frequency of 50 Hz.

To further evaluate latency performance, we apply for the same numbers of data sources (from 1× to 10×) with different combinations of Flink parallelisms and Kafka partitions. Fig. 10(a) illustrates that more Flink parallelisms could provide a relatively greater performance when handling a large workload of stream processing. For example, six parallel Flink TMs can deal with 60 PMU data sources with the processing latency around 15 ms. The reason is that more parallel Flink TMs can fetch more PMU data streams from Kafka partitions in time and perform stream processing more efficiently. Moreover, we observe that when the numbers of Flink TMs and Kafka partitions match, for example, six Flink workers and six Kafka partitions, the streaming processing latency decreases.

On the other hand, to compare KFIML with the *structured streaming* in Spark framework utilized in [23], we run Spark master and workers in containers orchestrated by Kubernetes and keep the same computing environment (i.e., Java heap size). Through experiments, the default Spark Structured Streaming generates a high latency about 1–2 s because it is based on traditional micro-bath processing engine. The *continuous processing* is a new execution mode in Structured Streaming to achieve streaming processing without the micro-bath processing engine. For the sake of fairness, we compare the *continuous processing* mode with our structured KFIML platform under the same set of Flink parallelisms and Kafka partitions. The experimental results shown in Fig. 10 indicate that the average real-time power computation latency of *continuous processing* is relatively higher.

To evaluate the efficiency of scaled structured KFIML, we double the software components including EMQ brokers, Kafka brokers, Flink TMs, and Influxdb running on distributed fog nodes, and then compare the average CPU overhead of each server when dealing with the increasing amount of PMU data (from 1× to 8×). Fig. 11 illustrates that the average CPU usages of the worker nodes and IoT broker servers increase almost linearly with the increasing number of data sources. Fig. 11(b) shows that scaled software containers at the worker nodes consume about twice of distributed CPU resources compared with the original software specification. The result indicates that the applications on the KFIML platform can take advantage of limited fog computing resources by scaling up and have the ability to deal with increased overloads of processing multisource data streams.

We further compare the average scaling time of Flink TMs and ML applications with and without the use of the

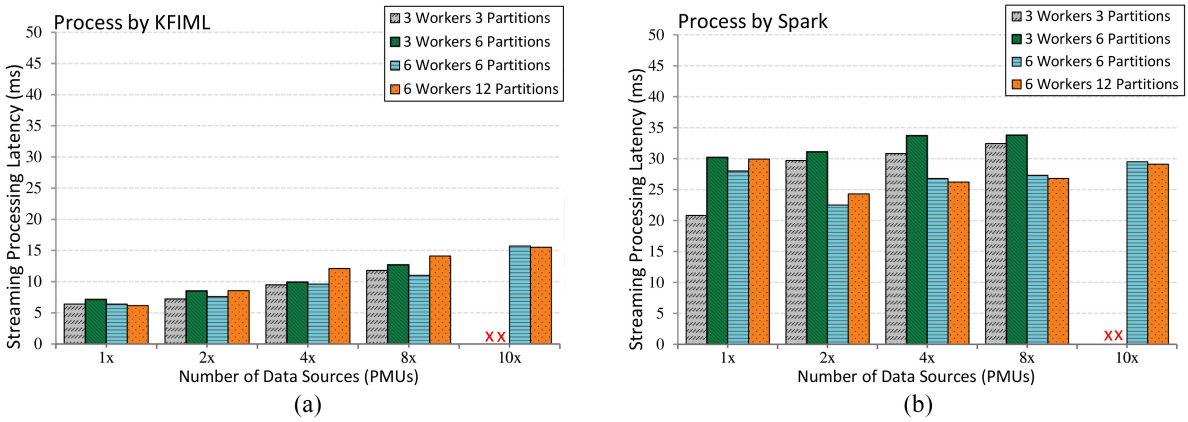
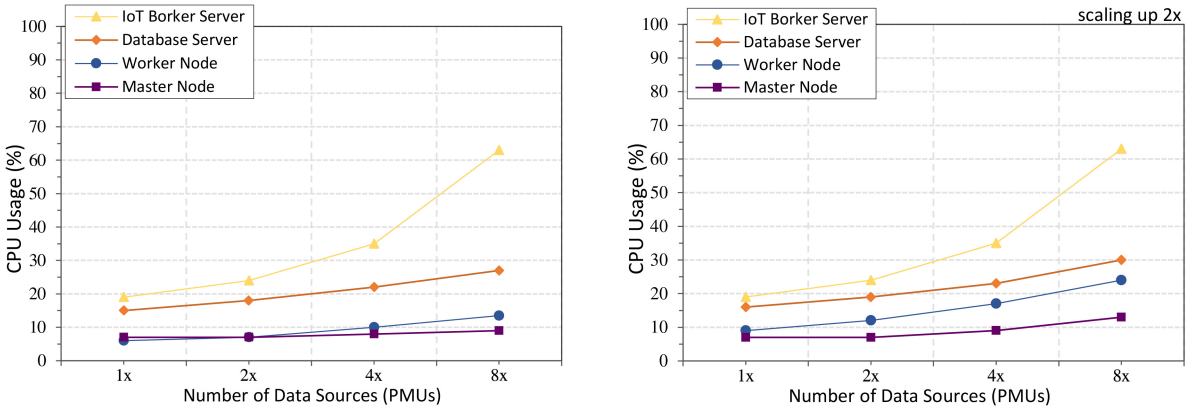


Fig. 10. Comparison of streaming processing latency between KFIML and Spark Structured Streaming with increased data sources. (a) Latency performance of KFIML with different parallelisms. (b) Latency performance of Spark with different parallelisms.



(a) CPU performance of standard KFIML platform

(b) CPU performance of scaled-up KFIML platform

Fig. 11. CPU utilization of KFIML with different scaling replicas when handling increased data sources. (a) CPU performance of standard KFIML platform. (b) CPU performance of scaled-up KFIML platform.

Kubernetes orchestration system. As illustrated in Fig. 12, the overall scaling latency of the default standalone mode is greater than that of KFIML with Kubernetes. This has something to do with the dynamic pod scheduling mechanism in Kubernetes and the isolation property of containers. Kubernetes provides a more elastic and automatic process to scale containerized software on accessible cluster resources, regardless of the local executing environment (e.g., Java 8) on the host operating system such as Ubuntu. When scaling containers of KFIML, the Kubernetes API server receives the scaling request, obtains the specific number of replicas, and creates a required number of containers on distributed fog servers in parallel. In contrast, now that the traditional standalone mode of scaling software on distributed servers does not depend on containers, we need to sequentially update required replicas. The above experimental results verify that our KFIML is marked by remarkable efficiency and exceptional scalability against the increased IoT data streams.

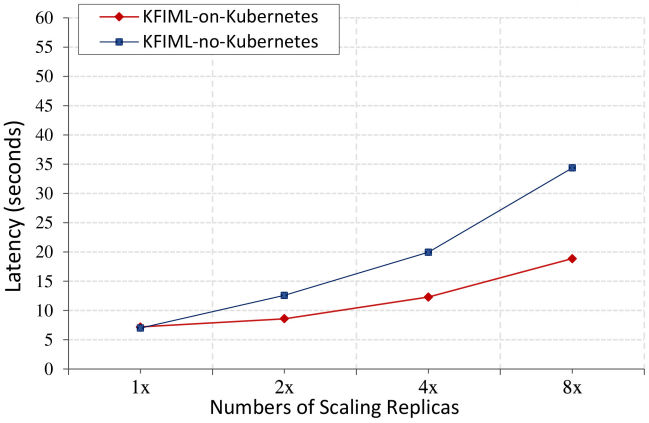


Fig. 12. Comparison of scaling time for multiple replicas with and without the use of the Kubernetes.

VII. Conclusion

To handle the massive onsite IoT data streams from IoE applications, this article proposes a five-tier fog computing platform based on a comprehensive IoT data processing workflow. A heterogeneous and clustered testbed, composed of four-type fog servers, is implemented to verify its feasibility. We then select the EPFL smart grid project as a use case to test the functionality and efficiency of the proposed testbed. The platform adopts the advanced big data framework,

Flink, to perform both streaming processing and statistical analysis of real-time high-velocity data streams. In addition, a specified LSTM-based ML pipeline, which comprises online training, model validation, and online prediction, is deployed with Tensorflow to predict the future power parameters based on onsite power statistical results. Furthermore, we utilize the light-weight orchestration system, namely, Kubernetes, to scale and manage these container-based frameworks. The experimental results show that the window size of big data stream processing and the learning rate of the online training process should be properly selected for each data set. The results also demonstrate that the proposed KFIML platform is efficient and scalable when handling multisource and real-time IoT data streams.

In our future work, we will extend our platform to support big data processing and ML-based analysis of multisource IoT data streams from more complicated IoT scenarios. In addition, our proposed LSTM-based ML pipeline for online prediction has limited abilities to handle large numbers of data sources. Therefore, developing a more comprehensive ML model for online predicting multisource onsite IoT data will be a crucial future direction. Some emerging projects, such as Flink AI Flow [39], which support running ML-based applications on the big data framework, also deserve further investigation and comparison with our platform.

REFERENCES

[1] S. Maghsudi and M. Davy, "Computational models of human decision-making with application to the Internet of Everything," *IEEE Wireless Commun.*, vol. 28, no. 1, pp. 152–159, Feb. 2021.

[2] B. de Mello Alencar, R. A. Rios, C. J. L. de Santana, and C. V. S. Prazeres, "Fot-stream: A fog platform for data stream analytics in IoT," *Comput. Commun.*, vol. 164, pp. 77–87, Dec. 2020.

[3] R. Mahmud, A. N. Toosi, K. Ramamohanarao, and R. Buyya, "Context-aware placement of industry 4.0 applications in fog computing environments," *IEEE Trans. Ind. Informat.*, vol. 16, no. 11, pp. 7004–7013, Nov. 2020.

[4] S. Chen, X. Zhu, H. Zhang, C. Zhao, G. Yang, and K. Wang, "Efficient privacy preserving data collection and computation offloading for fog-assisted IoT," *IEEE Trans. Sustain. Comput.*, vol. 5, no. 4, pp. 526–540, Oct.–Dec. 2020.

[5] T.-A. N. Abdali, R. Hassan, A. H. M. Aman, and Q. N. Nguyen, "Fog computing advancement: Concept, architecture, applications, advantages, and open issues," *IEEE Access*, vol. 9, pp. 75961–75980, 2021.

[6] E. Baccarelli, P. G. V. Naranjo, M. Scarpiniti, M. Shojafar, and J. H. Abawajy, "Fog of everything: Energy-efficient networked computing architectures, research challenges, and a case study," *IEEE Access*, vol. 5, pp. 9882–9910, 2017.

[7] B. Tang *et al.*, "Incorporating intelligence in fog computing for big data analysis in smart cities," *IEEE Trans. Ind. Informat.*, vol. 13, no. 5, pp. 2140–2150, Oct. 2017.

[8] T. Islam and M. M. A. Hashem, "A big data management system for providing real time services using fog infrastructure," in *Proc. IEEE Symp. Comput. Appl. Ind. Electron. (ISCAIE)*, 2018, pp. 85–89.

[9] P. Pop *et al.*, "The FORA fog computing platform for industrial IoT," *Inf. Syst.*, vol. 98, May 2021, Art. no. 101727.

[10] L. Hu *et al.*, "A dynamic pyramid tilling method for traffic data stream based on Flink," *IEEE Trans. Intell. Transp. Syst.*, early access, Mar. 4, 2021, doi: 10.1109/TITS.2021.3060576.

[11] A. Khochare, A. Krishnan, and Y. Simmhan, "A scalable platform for distributed object tracking across a many-camera network," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 6, pp. 1479–1493, Jun. 2021.

[12] P. Garefalakis, K. Karanasos, and P. Pietzuch, "Neptune: Scheduling suspendable tasks for unified stream/batch applications," in *Proc. ACM Symp. Cloud Comput.*, New York, NY, USA, 2019, pp. 233–245.

[13] A. Akbar, A. Khan, F. Carrez, and K. Moessner, "Predictive analytics for complex IoT data streams," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1571–1582, Oct. 2017.

[14] K. H. Abdulkareem *et al.*, "A review of fog computing and machine learning: Concepts, applications, challenges, and open issues," *IEEE Access*, vol. 7, pp. 153123–153140, 2019.

[15] A. T. Eseye, M. Lehtonen, T. Tukia, S. Uimonen, and R. J. Millar, "Machine learning based integrated feature selection approach for improved electricity demand forecasting in decentralized energy systems," *IEEE Access*, vol. 7, pp. 91463–91475, 2019.

[16] J. Chen *et al.*, "A parallel random forest algorithm for big data in a spark cloud computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 919–933, Apr. 2017.

[17] M. C. Pegalajar, L. G. B. Ruiz, M. P. Cuéllar, and R. Rueda, "Analysis and enhanced prediction of the Spanish electricity network through big data and machine learning techniques," *Int. J. Approx. Reason.*, vol. 133, pp. 48–59, Jun. 2021.

[18] N. D. Nguyen, L.-A. Phan, D.-H. Park, S. Kim, and T. Kim, "ElasticFog: Elastic resource provisioning in container-based fog computing," *IEEE Access*, vol. 8, pp. 183879–183890, 2020.

[19] K. Kaur, T. Dhand, N. Kumar, and S. Zeadally, "Container-as-a-service at the edge: Trade-off between energy efficiency and service availability at fog nano data centers," *IEEE Wireless Commun.*, vol. 24, no. 3, pp. 48–56, Jun. 2017.

[20] N. Zhao *et al.*, "Large-scale analysis of docker images and performance implications for container storage systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 4, pp. 918–930, Apr. 2021.

[21] P. Kayal, "Kubernetes in fog computing: Feasibility demonstration, limitations and improvement scope : Invited paper," in *Proc. IEEE 6th World Forum Internet Things (WF-IoT)*, 2020, pp. 1–6.

[22] K. Kaur, S. Garg, G. Kaddoum, S. H. Ahmed, and M. Atiquzzaman, "KEIDS: Kubernetes-based energy and interference driven scheduler for industrial IoT in edge-cloud ecosystem," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4228–4237, May 2020.

[23] S. Nguyen, Z. Salcic, X. Zhang, and A. Bisht, "A low-cost two-tier fog computing testbed for streaming IoT-based applications," *IEEE Internet Things J.*, vol. 8, no. 8, pp. 6928–6939, Apr. 2021.

[24] J. He, J. Wei, K. Chen, Z. Tang, Y. Zhou, and Y. Zhang, "Multitier fog computing with large-scale IoT data analytics for smart cities," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 677–686, Apr. 2018.

[25] X. Wang *et al.*, "A fog-based recommender system," *IEEE Internet Things J.*, vol. 7, no. 2, pp. 1048–1060, Feb. 2020.

[26] A. Yassine, S. Singh, M. S. Hossain, and G. Muhammad, "IoT big data analytics for smart homes with fog and cloud computing," *Future Gener. Comput. Syst.*, vol. 91, pp. 563–573, Feb. 2019.

[27] R. Jaiswal, A. Chakravorty, and C. Rong, "Distributed fog computing architecture for real-time anomaly detection in smart meter data," in *Proc. IEEE 6th Int. Conf. Big Data Comput. Serv. Appl. (BigDataService)*, 2020, pp. 1–8.

[28] N. F. Omran, S. F. Abd-el Ghany, H. Saleh, A. Nabil, and A. M. Khalil, "Breast cancer identification from patients' tweet streaming using machine learning solution on spark," *Complexity,* vol. 2021, p. 12, Jan. 2021, doi: 10.1155/2021/6653508.

[29] "EMQX Broker Documentation: Rule Engine." 2021. [Online]. Available: https://docs.emqx.io/en/broker/v4.3/rule/rule-engine.html

[30] "Kubernetes: StatefulSet." 2020. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/

[31] "Flink Documentation: Kubernetes Setup." 2021. [Online]. Available: https://ci.apache.org/projects/flink/flink-docs-release-1.13/docs/deployment/resource-providers/standalone/kubernetes/

[32] M. Cui, J. Wang, J. Tan, A. R. Florita, and Y. Zhang, "A novel event detection method using PMU data with high precision," *IEEE Trans. Power Syst.*, vol. 34, no. 1, pp. 454–466, Jan. 2019.

[33] X. Wang, D. Shi, J. Wang, Z. Yu, and Z. Wang, "Online identification and data recovery for PMU data manipulation attack," *IEEE Trans. Smart Grid*, vol. 10, no. 6, pp. 5889–5898, Nov. 2019.

[34] Z. Yang, N. Chen, Y. Chen, and N. Zhou, "A novel PMU fog based early anomaly detection for an efficient wide area PMU network," in *Proc. IEEE 2nd Int. Conf. Fog Edge Comput. (ICFEC)*, 2018, pp. 1–10.

[35] "EPFL Campus PMU Data Set Switzerland." 2019. [Online]. Available: https://bigdata.seas.gwu.edu/data-set-20-epfl-campus-pmu-data-set/

[36] "MQTT 5 Specification." 2019. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf

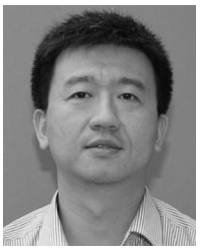
[37] "Flink Documentation: Apache Kafka Connector." 2021. [Online]. Available: https://ci.apache.org/projects/flink/flink-docs-release-1.13/docs/connectors/datastream/kafka/
[38] W. Zhang *et al.*, "LSTM-based analysis of industrial IoT equipment," *IEEE Access*, vol. 6, pp. 23551–23560, 2018.
[39] "Flink AI Flow." 2021. [Online]. Available: https://github.com/alibaba/flink-ai-extended/tree/master/flink-ai-flow

Ziyu Wan received the B.S. degree in communication engineering from the School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing, China, in 2020. He is currently pursuing the M.S. degree with the College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou, China.

His current research interests mainly include Internet of Things, and edge and cloud computing.

Zheng Zhang received the B.S. degree in communication engineering from the School of Telecommunications Engineering, Xidian University, Xi'an, China, in 2020. He is currently pursuing the M.S. degree with the College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou, China.

His current research interests include signal processing, deep learning, and data analysis.



Rui Yin (Senior Member, IEEE) received the B.S. degree in computer engineering from Yanbian University, Yanji, China, in 2001, the M.S. degree in computer engineering from the University of KwaZulu-Natal, Durban, South Africa, in 2006, and the Ph.D. degree in information and electronic engineering from Zhejiang University, Hangzhou, China, in 2011.

From March 2011 to June 2013, he was a Research Fellow with the Department of Information and Electronic Engineering, Zhejiang University. He is currently a Professor with the School of Information and Electrical Engineering, Zhejiang University City College, Hangzhou, and a joint Honorary Research Fellow with the School of Electrical, Electronic and Computer Engineering, University of KwaZulu-Natal. His research interests mainly focus on radio resource management in LTE unlicensed, millimeter-wave cellular wireless networks, HetNet, cooperative communications, massive MIMO, optimization theory, game theory, and information theory.

Prof. Yin regularly serves as the Technical Program Committee boards of prominent IEEE conferences, such as ICC, GLOBECOM, and PIMRC and chairs some of their technical sessions and reviewer for IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, IEEE TRANSACTIONS ON COMMUNICATIONS, IEEE WIRELESS COMMUNICATIONS, *IEEE Communications Magazine*, IEEE NETWORK, and IEEE TRANSACTIONS ON SIGNAL PROCESSING.

Guanding Yu (Senior Member, IEEE) received the B.E. and Ph.D. degrees in communication engineering from Zhejiang University, Hangzhou, China, in 2001 and 2006, respectively.

He joined Zhejiang University in 2006, where he is currently a Professor with the College of Information Science and Electronic Engineering. From 2013 to 2015, he was also a Visiting Professor with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA. His research interests include 5G communications and networks, integrated sensing and communications, mobile-edge computing/learning, and machine learning for wireless networks.

Dr. Yu received the 2016 IEEE ComSoc Asia–Pacific Outstanding Young Researcher Award. He has served as a Guest Editor for *IEEE Communications Magazine* Special Issue on Full-Duplex Communications, an Editor for IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS Series on Green Communications and Networking and Series on Machine Learning in Communications and Networks, an Editor for IEEE WIRELESS COMMUNICATIONS LETTERS, IEEE TRANSACTIONS ON GREEN COMMUNICATIONS AND NETWORKING, and IEEE ACCESS, and a Lead Guest Editor for *IEEE Wireless Communications Magazine* Special Issue on LTE in Unlicensed Spectrum. He regularly sits on the Technical Program Committee boards of prominent IEEE conferences, such as ICC, GLOBECOM, and VTC. He has served as a Symposium Co-Chair for IEEE Globecom 2019 and a Track Chair for IEEE VTC 2019'Fall.