INVITED PAPER

A Survey on Edge Computing Systems and Tools

By Fang Liu, Guoming Tang, Youhuizi Li, *Member IEEE*, Zhiping Cai, *Member IEEE*, Xingzhou Zhang, and Tongqing Zhou

ABSTRACT | Driven by the visions of Internet of Things and 5G communications, the edge computing systems integrate computing, storage, and network resources at the edge of the network to provide computing infrastructure, enabling developers to quickly develop and deploy edge applications. At present, the edge computing systems have received widespread attention in both industry and academia. To explore new research opportunities and assist users in selecting suitable edge computing systems for specific applications, this survey paper provides a comprehensive overview of the existing edge computing systems and introduces representative projects. A comparison of open-source tools is presented according to their applicability. Finally, we highlight energy efficiency and deep learning optimization of edge computing systems. Open issues for analyzing and designing an edge computing system are also studied in this paper.

KEYWORDS | Deep learning optimization; edge computing systems; energy efficiency; open-source tools; survey.

I. INTRODUCTION

In the post-Cloud era, the proliferation of Internet of Things (IoT) and the popularization of 4G/5G gradually change the public's habit of accessing and processing data and challenge the linearly increasing capability of cloud computing. Edge computing is a new computing paradigm with data processed at the edge of the network. Promoted by the fast-growing demand and interest in this area, the edge computing systems and tools are blooming, even though some of them may not be popularly used right now.

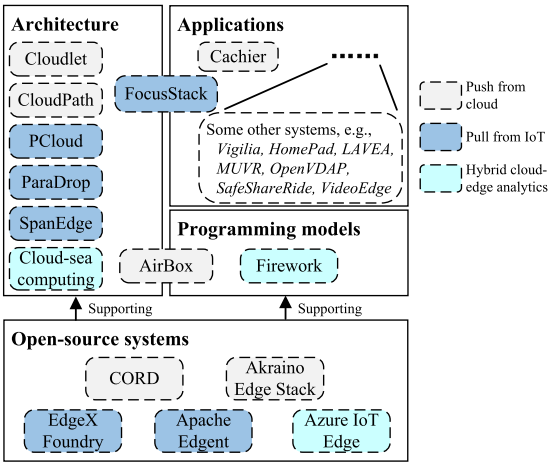


Fig. 1. *Categorization of edge computing systems.*

There are many classification perspectives to distinguish different edge computing systems. To figure out why edge computing appears as well as its necessity, we pay more attention to the basic motivations. Specifically, based on different design demands, existing edge computing systems can roughly be classified into three categories, together yielding innovations on system architecture, programming models, and various applications, as shown in Fig. 1.

1) *Push From Cloud:* In this category, cloud providers push services and computation to the edge in order to leverage locality, reduce response time, and improve user experience. Representative systems include Cloudlet, Cachier, AirBox, and CloudPath. Many traditional cloud computing service providers are actively pushing cloud services closer to users, shortening the distance between customers and

Manuscript received February 6, 2019; revised April 13, 2019; accepted May 23, 2019. This work was supported in part by the National Key Research and Development Program of China under Grant 2016YFB1000302 and in part by the National Natural Science Foundation of China under Grant 61433019. *(Corresponding authors: Fang Liu; Zhiping Cai.)*

F. Liu is with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510006, China (e-mail: liufang25@mail.sysu.edu.cn).

G. Tang is with the Key Laboratory of Science and Technology on Information System Engineering, National University of Defense Technology, Changsha 410073, China (e-mail: gmtang@nudt.edu.cn).

Y. Li is with the School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou 310002, China.

Z. Cai and **T. Zhou** are with the College of Computer, National University of Defense Technology, Changsha 410073, China (e-mail: zpcai@nudt.edu.cn; zhoutongqing1991@163.com).

X. Zhang is with the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China.

Digital Object Identifier 10.1109/JPROC.2019.2920341

cloud computing, so as not to lose market to mobile edge computing (MEC). For example, Microsoft launched AzureStack in 2017, which allows cloud computing capabilities to be integrated into the terminal, and data can be processed and analyzed on the terminal device.

2) *Pull From IoT:* IoT applications pull services and computation from the faraway cloud to the near edge to handle the huge amount of data generated by IoT devices. Representative systems include PCloud, ParaDrop, FocusStack, and SpanEdge. Advances in embedded systems-on-a-chip (SoCs) have given rise to many IoT devices that are powerful enough to run embedded operating systems and complex algorithms. Many manufacturers integrate machine learning (ML) and even deep learning capabilities into IoT devices. Utilizing edge computing systems and tools, IoT devices can effectively share computing, storage, and network resources while maintaining a certain degree of independence.
3) *Hybrid Cloud-Edge Analytics:* The integration of advantages of cloud and edge provides a solution to facilitate both global optimal results and minimum response time in modern advanced services and applications. Representative systems include Firework and Cloud-Sea computing systems. Such edge computing systems utilize the processing power of IoT devices to filter, preprocess, and aggregate IoT data while employing the power and flexibility of cloud services to run complex analytics on those data. For example, Alibaba Cloud launched its first IoT edge computing product, LinkEdge, in 2018, which expands its advantages in cloud computing, big data, and artificial intelligence (AI) to the edge to build a cloud/edge-integrated collaborative computing system; Amazon released Amazon Web Services (AWS) Greengrass in 2017, which can extend AWS seamlessly to devices so that devices can perform local operations on the data they generate, while data are transferred to the cloud for management, analysis, and storage.

From a research point of view, this paper gives a detailed introduction to the distinctive ideas and model abstractions of the aforementioned edge computing systems. Note that the three categories are presented to clearly explain the necessity of edge computing, and the classification is not the main line of this paper. Specifically, we review systems designed for architecture innovation first, then introduce those for programming models and applications (in Section II). In addition, some recent efforts for specific application scenarios are also studied.

While we can find a lot of systems using edge computing as the building block, there still lacks standardization to such a paradigm. Therefore, a comprehensive and coordinated set of foundational open-source systems/tools is also needed to accelerate the deployment of IoT and edge computing solutions. Some open-source edge computing projects have been launched recently [e.g., Central Office Re-architected as a Datacenter (CORD)]. As shown in Fig. 1, these systems can support the design of both architecture and programming models with useful application programming interfaces (APIs). We review these open-source systems with a comparative study on their characteristics (in Section III).

When designing the edge computing system described above, energy efficiency is always considered as one of the major concerns as the edge hardware is energy-restriction. Meanwhile, the increasing number of IoT devices is bringing the growth of energy-hungry services. Therefore, we also review the energy-efficiency-enhancing mechanisms adopted by the state-of-the-art edge computing systems from the three-layer paradigm of edge computing (in Section IV).

In addition to investigations from the system view, we also look into the emerging techniques in the edge computing system from the application view. Recently, deep learning-based AI applications are widely used and offloading the AI functions from the cloud to the edge is becoming a trend. However, deep learning models are known for being large and computationally expensive. Traditionally, many systems and tools are designed to run deep learning models efficiently on the cloud. As the multilayer structure of deep learning, it is appropriate for edge computing paradigm, and more of its functions can be offloaded to the edge. Accordingly, this paper also studies the new techniques recently proposed to support the deep learning models at the edge (in Section V).

Our main contributions in this paper are as follows.

1) Reviewing existing systems and open-source projects for edge computing by categorizing them from their design demands and innovations. We study the targets, architecture, characteristics, and limitations of the systems in a comparative way.
2) Investigating the energy-efficiency-enhancing mechanism for edge computing from the view of the cloud, the edge servers, and the battery-powered devices.
3) Studying the technological innovations dedicated to deploying deep learning models on the edge, including systems and toolkits, packages, and hardware.
4) Identifying challenges and open research issues of edge computing systems, such as mobility support, multiuser fairness, and privacy protection.

We hope this effort will inspire further research on edge computing systems. The contents and their organization in this paper are shown in Fig. 2. Besides the major four building blocks (Sections II–V), we also give a list of open issues for analyzing and designing an edge computing system in Section VI. This paper is concluded in Section VII.

II. EDGE COMPUTING SYSTEMS AND TOOLS

In this section, we review edge computing systems and tools presenting architecture innovations, programming

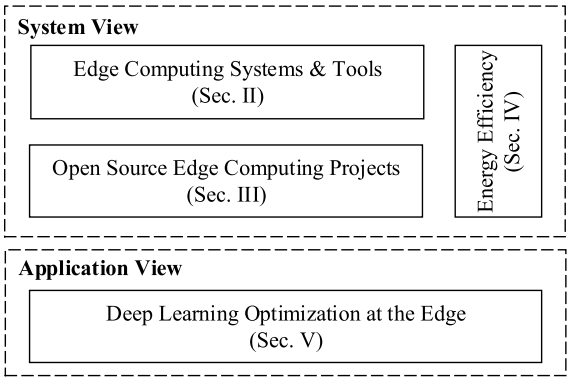


Fig. 2. ***Major building blocks and organization of this survey paper.***

models, and applications, respectively. For each part, we introduce work under the "push," "pull," and "hybrid" demand in turn.

A. Cloudlet

In 2009, Carnegie Mellon University (CMU) proposed the concept of Cloudlet [1], and the Open Edge computing initiative was also evolved from the Cloudlet project [2]. Cloudlet is a trusted, resource-rich computer or cluster of computers that are well-connected to the Internet and available to nearby mobile devices. It upgrades the original two-tier architecture "Mobile Device-Cloud" of the mobile cloud computing to a three-tier architecture "Mobile Device-Cloudlet-Cloud." Meanwhile, Cloudlet can also serve users like an independent cloud, making it a "small cloud" or "data center (DC) in a box." Although the Cloudlet project is not proposed and launched in the name of edge computing, its architecture and ideas fit those of the edge computing and thus can be regarded as an edge computing system.

The Cloudlet is in the middle layer of the three-tier edge computing architecture and can be implemented on a personal computer, low-cost server, or small cluster. It can be composed of a single machine or small clusters consisting of multiple machines. Like WiFi service access points, a Cloudlet can be deployed at a convenient location (such as a restaurant, a cafe, or a library). Multiple Cloudlets may form a distributed computing platform, which can further extend the available resources for mobile devices [3]. As the Cloudlet is just one hop away from the users' mobile devices, it improves the QoS with low communication delay and high bandwidth utilization.

In detail, Cloudlet has three main features as follows.

1) *Soft State:* Cloudlet can be regarded as a small cloud computing center located at the edge of the network. Therefore, as the server end of the application, the Cloudlet generally needs to maintain state information for interacting with the client. However, unlike Cloud, Cloudlet does not maintain long-term state information for interactions, but only temporarily caches some state information. This reduces much of the burden of Cloudlet as a lightweight cloud.
2) *Rich Resources:* Cloudlet has sufficient computing resources to enable multiple mobile users to offload computing tasks to it. In addition, Cloudlet also has stable power supply so it does not need to worry about energy exhaustion.
3) *Close to Users:* Cloudlets are deployed at those places where both network distance and physical distance are short to the end user, making it easy to control the network bandwidth, delay, and jitter. In addition, the physical proximity ensures that the Cloudlet and the user are in the same context (e.g., the same location), based on which customized services (e.g., the location-based service) could be provided.

To further promote Cloudlet, CMU built up an open edge computing alliance, with Intel, Huawei, and other companies [2], to develop standardized APIs for Cloudlet-based edge computing platforms. Currently, the alliance has transplanted OpenStack to the edge computing platform, which enables distributed Cloudlet control and management via the standard OpenStack APIs [4]. With the recent development of the edge computing, the Cloudlet paradigm has been widely adopted in various applications, e.g., cognitive assistance system [5], [6], IoT data analysis [7], and hostile environments [8].

Unlike the cloud, cloudlets are deployed on the edge of the network and serve only nearby users. Cloudlet supports application mobility, allowing devices to switch service requests to the nearest cloudlet during the mobile process. As shown in Fig. 3, Cloudlet supports for application mobility relying on three key steps.

1) *Cloudlet Discovery:* Mobile devices can quickly discover the available Cloudlets around them and choose the most suitable one to offload tasks.
2) *Virtual Machine (VM) Provisioning:* Configuring and deploying the service VM that contains the server code on the cloudlet so that it is ready to be used by the client.
3) *VM Handoff:* Migrating the VM running the application to another cloudlet.

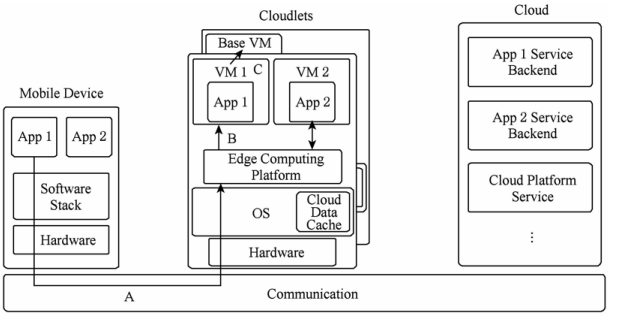


Fig. 3. ***Cloudlet component overview and functions that support application mobility. A: cloudlet discovery. B: VM provisioning. C: VM handoff.***

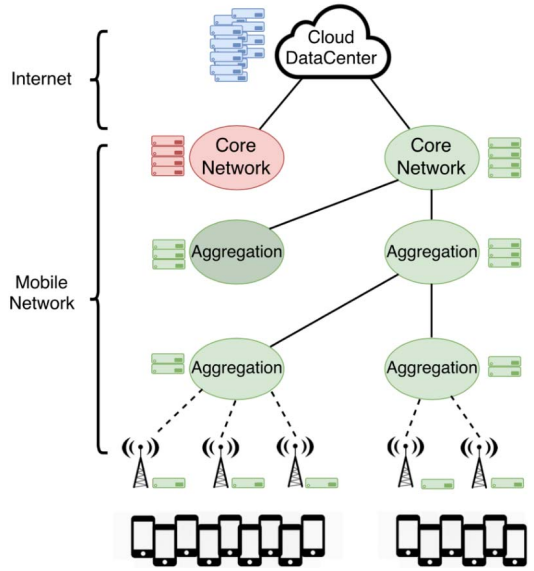


Fig. 4. ***CloudPath architecture [9].***

B. CloudPath

CloudPath [9] is an edge computing system proposed by the University of Toronto. In such a system, diverse resources such as computing and storage are provided along the path from the user device to the cloud DC. It supports on-demand allocation and dynamic deployment of the multilevel architecture. The main idea of CloudPath is to implement the so-called "path computing," such that it can reduce the response time and improve the bandwidth utilization, compared with the conventional cloud computing.

As illustrated in Fig. 4, the bottom layer of CloudPath is user devices, and the top layer is the cloud computing DC. The system reassigns those tasks of the DCs along the path (for path computing) to support different types of applications, such as IoT data aggregation, data caching services, and data processing services. Developers can select an optimal hierarchical deployment plan for their services by considering the factors such as cost, delay, resource availability, and geographic coverage. Path computing builds a multi-tiered architecture, and from the top (traditional DC) to the bottom (user terminal equipment), the device capability becomes weaker, while the number of devices gets larger. On the premise of a clear separation of computing and states, CloudPath extends the abstract shared storage layer to all DC nodes along the path, which reduces the complexity of third-party application development and deployment and, meanwhile, keeps the RESTful development style.

The CloudPath application consists of a set of short-cycle and stateless functions that can be quickly instantiated at any level of the CloudPath framework. Developers either tag functions to specify where (such as edges, cores, clouds, etc.) their codes run, or tag performance requirements (such as response latency) to estimate the running location. CloudPath does not migrate a running function/module but supports service mobility by stopping the current instance and re-starting a new one at the expected location. Each CloudPath node usually consists of the following six modules.

1) *PathExecute:* It implements a serverless cloud container architecture that supports lightweight stateless application functions/modules.
2) *PathStore:* It provides a distributed eventual consistent storage system that transparently manages application data across nodes.
3) *PathRoute:* It transmits the request to the most appropriate CloudPath node, according to the information such as user's location in the network, application preferences, or system status.
4) *PathDeploy:* It dynamically deploys and removes applications on CloudPath nodes based on application preferences and system policies.
5) *PathMonitor:* It provides real-time monitoring and historical data analysis function to applications and CloudPath nodes. It collects the metrics of other CloudPath modules on each node through the PathStore and presents the data to users using web pages.
6) *PathInit:* It is an initialization module in the top-level DC node and can be used to upload applications/services to the CloudPath.

C. PCloud

PCloud [10] integrates the edge computing and storage resources with those at the cloud to support seamless mobile services. The architecture of PCloud is

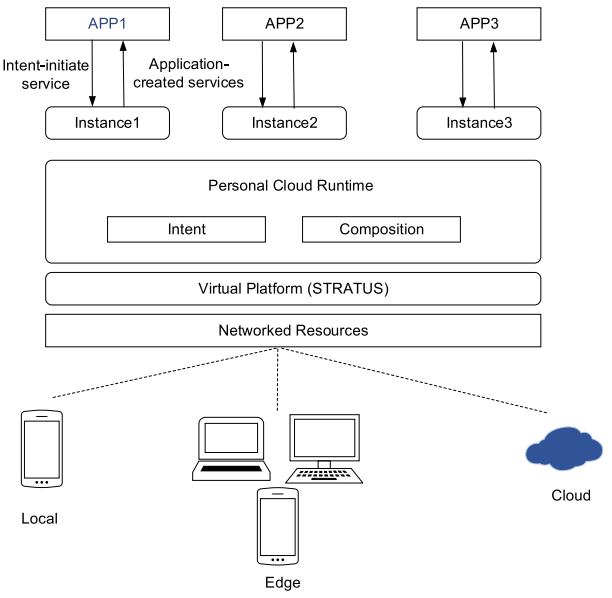


Fig. 5. ***PCloud architecture [10].***

shown in Fig. 5. Specifically, these resources are virtualized through a special virtualization layer named STRATUS [11] and form a distributed resource pool that can discover new resources and monitor resource changes. With the resource pool, the runtime mechanism is responsible for resource application and allocation. Through a resource description interface, the runtime mechanism selects and combines appropriate resources based on the requirements of specified applications. After the resources are combined, it generates a new instance to provide corresponding services for external applications, according to the resource access control policy. (Note that the computing resources of the newly generated instance may come from multiple devices, which is equivalent to one integrated computing device for the external applications.) Thus, an application program is actually a combination of services running on the PCloud instance. For example, a media player application can be a combination of the storage, decoding, and playing services. These services could be local or remote but are transparent to the applications. Furthermore, the PCloud system also provides basic system services, such as permission management and user data aggregation, to control the resources access of other users.

In the actual operation, the mobile application describes the required resources to the PCloud through interfaces. The PCloud will find out the optimal resource configuration by analyzing the description and the currently available resources and then generates an instance to provide corresponding services for the application. PCloud integrates edge resources with cloud resources so that they can complement each other. The abundant resources from the cloud can make up for the lack of computing and storage capabilities at the edge; meanwhile, due to the physical proximity, edge devices can provide low-latency services to the user that cloud cannot offer. In addition, PCloud also enhances the availability of the entire system and can choose alternate resources when encountering network and equipment failures.

D. ParaDrop

ParaDrop [12] is developed by the WiNGS Laboratory at the University of Wisconsin-Madison. It is an edge computing framework that makes the computing/storage resources close to mobile devices and data sources available to the third-party developers. Its goal is to bring intelligence to the network edge in a friendly way.

ParaDrop upgrades the existing access point to an edge computing system, which supports applications and services like a normal server. To isolate applications under the multitenacy scenario, ParaDrop leverages the lightweight container virtualization technique. Fig. 6 shows that the ParaDrop server (in the cloud) controls the deployment, starting, and deletion of the applications. It provides a group of APIs, via which the developer can monitor and manage the system resources and configure the running

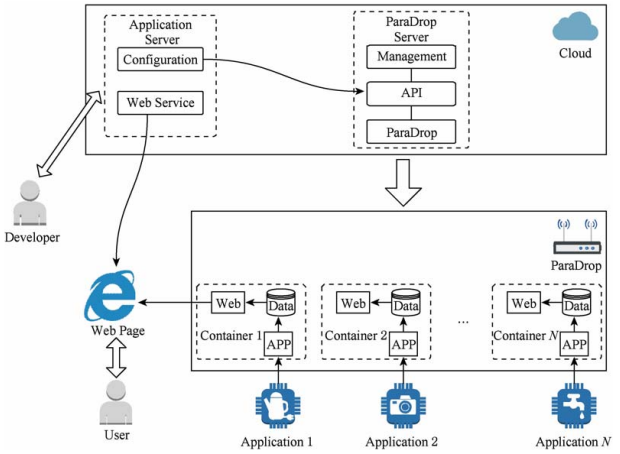


Fig. 6. ***ParaDrop system [12].***

environment. The web UI is also provided, through which the user can directly interact with the applications.

The design goals of ParaDrop include three aspects: multitenacy, efficient resource utilization, and dynamic application management. To achieve these goals, the container technology is applied to manage the multitenacy resources separately. As the resources of the edge devices are very limited, compared with the VM, the container consumes less resource and would be more suitable for delay sensitive and high I/O applications. Moreover, as applications running in the container, ParaDrop can easily control their startup and revocation.

ParaDrop is mainly used for IoT applications, especially IoT data analysis. Its advantages over traditional cloud system can be summarized as follows: 1) since sensitive data can be processed locally, it protects the users' privacy; 2) WiFi access point is only one hop away from the data source, leading to low network delay and stable connection; 3) only user requested data are transmitted to the equipment through the Internet, thus cutting down the total traffic amount and saving the bandwidth of the backbone network; 4) the gateway can obtain the location information of the edge devices through radio signals (e.g., the distance between devices and the location of the specific device), which facilitates the location-aware services; and 5) when edge devices cannot be connected to the Internet, the edge service can still work.

E. SpanEdge

Streaming processing is one important type of applications in edge computing, where data are generated by various data sources in different geographical locations and continuously transmitted in the form of streams. Traditionally, all raw data are transmitted over the WAN to the DC server, and stream processing systems, such as Apache Spark and Flink, are also designed and optimized for one centralized DC. However, this approach cannot effectively handle the huge data generated by a lot of devices at the

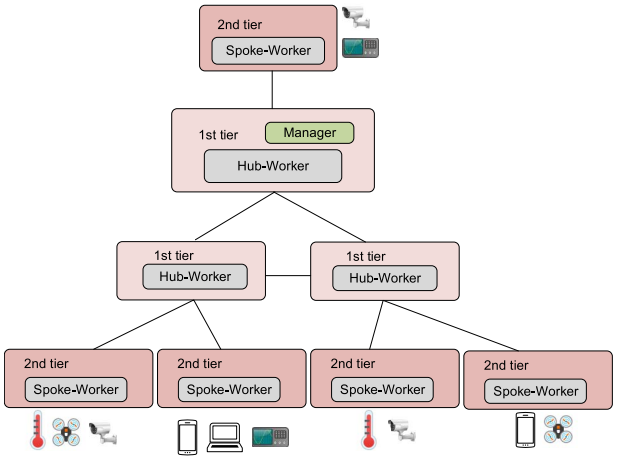


Fig. 7. ***SpanEdge architecture [13].***

edge of the network, and the situation is even worse when the applications require low latency and predictability. SpanEdge [13] is a research project of the Royal Institute of Technology in Sweden. It unifies the cloud central node and the near-edge central node, reduces network latency in WAN connections, and provides a programming environment that allows the program to run near the data source. Developers can focus on developing streaming applications without considering where the data sources are located and distributed.

The DC in SpanEdge is composed of two levels: the cloud DC is the first level and the edge DC (such as the operator's cloud, Cloudlet, or Fog) is the second level. Partial streaming processing tasks run on the edge central nodes to reduce latency and boost performance. SpanEdge uses the master–worker architecture (as shown in Fig. 7) with one manager and multiple workers. The manager collects the streaming processing requests and assigns tasks to the workers. Workers mainly consist of cluster nodes whose primary responsibility is to execute tasks. There are two types of workers: hub-worker (first level) and spoke-worker (second level). The network transmission overhead and latency are related to the geographical location and network connection status between workers. The communication in SpanEdge is also divided into a system management communication (worker–manager) and a data transmission communication (worker–worker). System management communication aims to schedule tasks between managers and workers, and data transmission communication takes care of the data flow in each task. Each worker has an agent that handles system management operations, such as sending and receiving management information, monitoring compute nodes to ensure that they are running normally, and periodically sending heartbeat messages to the manager to ensure immediate recovery when the task fails.

SpanEdge allows developers to divide the tasks into local ones and global ones. Local tasks should run on the node near the data source and provide only part of the required data; global tasks are responsible for further processing the results of local tasks and aggregating all results. SpanEdge creates a copy of the local task on each spoke-worker, which has all corresponding data sources. If the data are insufficient, the task is dispatched to the hub-worker. The global task runs on a separate hub-worker, and the scheduler selects the optimal hub-worker based on the network delay (i.e., the distance from the spoke-worker in the network topology).

F. Cloud-Sea Computing Systems

The Cloud-Sea Computing Systems project [14] is a main research thrust of the Next Generation Information and Communication Technology initiative [the National Institute of Computer Technology (NICT), China, initiative] and a 10-year strategic priority research initiative, launched by the Chinese Academy of Science in 2012. The NICT initiative aims to address the three major technology challenges in the coming Zettabyte era: 1) improving the performance per watt by 1000 times; 2) supporting more applications from the human-cyber-physical ternary computing; and 3) enabling transformative innovations in devices, systems, and applications while without polluting beneficial information technology ecosystems.

In the cloud-sea computing system, "cloud" refers to the datacenters and "sea" refers to the terminal side (the client devices, e.g., human-facing and physical world-facing subsystems). The design of the project can be depicted from three levels: the overall systems architecture level, the datacenter server and storage system level, and the processor chip level. The project contains four research components: a computing model called representational state transfer (REST) 2.0 that extends REST [15] architectural style of Web computing to cloud-sea computing, a three-tier storage system architecture capable of managing ZBs of data, a billion-thread datacenter server with high energy efficiency, and an elastic processor aiming at energy efficiency of one trillion operations per second per watt.

As shown in Fig. 8, the cloud-sea computing model includes sea-side functions and cloud-side functions.

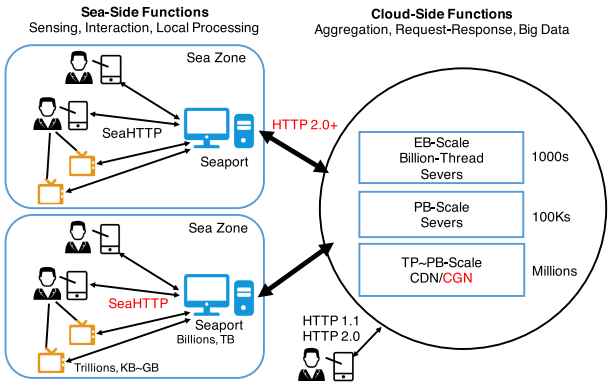


Fig. 8. ***Cloud-sea computing model.***

The sea zone is expanded from the traditional cloud client, e.g., a home, an office, and a factory manufacturing pipeline. There can be multiple client devices inside a sea zone, and each device can be a human facing or physical world facing. In a sea zone, there is a special device (like a home datacenter or a smart TV set) designated as the seaport for three purposes: 1) a gateway interfacing the sea zone to the cloud; 2) a gathering point of information and functionalities inside a sea zone; and 3) a shield protecting security and privacy of the sea zone. A device inside a sea zone does not communicate to the cloud directly, but through the seaport, either implicitly or explicitly. The SeaHTTP, a variant based on HTTP 2.0, is the widely used protocol in sea zones of the cloud-sea system to connect with the cloud.

The cloud-sea computing model has four distinct features.

1) *Ternary Computing via Sea Devices:* Human and physical world entities interface and collaborate with the cyberspace through the sea side. For example, users can leverage a smartphone application to read and control a sensor device in a home through an Internet application service.
2) *Cooperation With Locality:* A specific network computing system will partition its functions between the sea side and the cloud side. Sea-side functions include sensing, interaction, and local processing, while cloud-side functions include aggregations, request-response, and big data processing.
3) *Scalability to ZB and Trillion Devices:* This future Net will collectively need to support trillions of sea devices and to handle ZBs of data.
4) *Minimal Extension to Existing Ecosystems:* The REST 2.0 cloud-sea computing architecture attempts to utilize existing Web computing ecosystems as much as possible.

Overall, the cloud-sea computing model is proposed to migrate the cloud computing function to the sea side, and it focuses more on the devices at the "sea" side and the data at the "cloud" side. Typical edge computing is more generalized and may care about any intermediate computing resources and network resources between the "sea" and the "cloud." The research studies in cloud-sea computing (e.g., energy-efficient computing and elastic processor designing) are consultative for the edge computing.

G. Cachier and Precog

Cachier [16] and Precog [17] are two edge caching systems that were proposed by the researchers from CMU for image recognition. Recognition applications have strict response time requirements, while the computation is huge due to the model complexity and data set size. With edge computing, we can leverage the computing resource of the edge nodes to process the matching, so the network delay can be reduced. Considering that edge nodes mainly provide services to nearby users, the spatiotemporal characteristics of service requests can be leveraged. From the perspective of caching, the Cachier system proposes that edge nodes can be used as "computable cache devices," which can cache recognition results, reduce matching data set size, and improve response time. By analyzing the response delay model and requests' characteristics, the system can dynamically adjust the size of the data set on the edge nodes according to the environment factors, thus ensuring optimal response time.

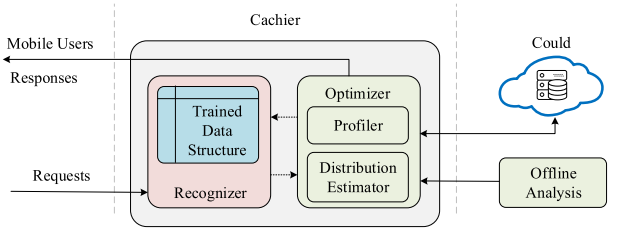


Fig. 9. ***Cachier system [16].***

Fig. 9 shows that Cachier consists of the recognizer module, the optimizer module, and the offline analysis module. The recognizer module is responsible for analyzing and matching the received figures according to the cached training data and model. If there is a match, the result will be directly returned to the user; otherwise, the figure will be transmitted to the cloud for recognition. The distribution estimator in the optimizer module can use the maximum *a posteriori* estimation to predict the request distribution. Second, given the classification algorithm and training data set, the cache searching delay and the cache accuracy can also be calculated by the offline analysis module. At last, the profiler submodule is responsible for estimating the network delay and cloud latency incurred by cache misses. It measures and records the delay under corresponding distance in real time and uses the moving average filter to remove the noise data. By taking such information into the delay expectation time model, Cachier is able to calculate the optimal cache size and then adjusts the cache on the edge nodes accordingly.

Precog is an extension of Cachier. The cached data are not only on the edge nodes but also on the end devices that are used for selection calculation in order to reduce the image migration between the edge nodes and the cloud. Based on the prediction model, Precog prefetches some of the trained classifiers, uses them to recognize the images, and cooperates with the edge nodes to efficiently complete the tasks. Precog pushes computing capability to the end device and leverages the locality and selectivity of the user requests to reduce the last mile delay. For the end device, if the cache system uses the same cache replacement policy as the edge node applies, it will lead to a large number of forced misses. For a single user, the device only has the request information of the user, and usually, the user will not identify the same object multiple times in the near future. Therefore, Precog constructs a Markov model using the request information from the edge node to describe

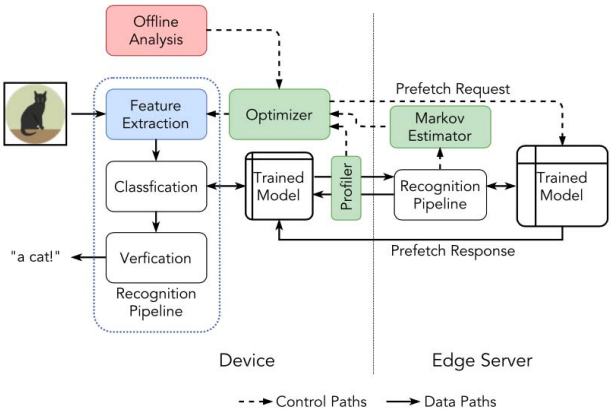


Fig. 10. ***Precog system [17].***

the relationship among the identified objects and then predicts the potential future requests. Precog improves the delay expectation time model by considering the network state, device capabilities, and the prediction information from edge nodes. The system architecture of Precog is illustrated in Fig. 10. We can see that it is mainly composed of the feature extraction module, the offline analysis module, the optimizer module, and the profiler module. The edge node is mainly responsible for the construction of the Markov model. Based on the offline analysis results, the Markov model prediction results, and the network information provided by the profiler module, the optimizer module determines the number of features to be extracted as well as the data to be cached on the end device.

H. FocusStack

FocusStack [18] is developed by AT&T Labs, which supports the deployment of complex applications on a variety of potential IoT edge devices. Although the resources, such as computing, power consumption, and connectivity, are limited on edge devices, they have the nature of mobility. Hence, it is very important to have a system that can discover and organize available edge resources. FocusStack is such a system that can discover a set of edge devices with sufficient resources, deploy applications, and run them accordingly. Thus, the developers can focus more on the design of applications than on how to find and track edge resources.

FocusStack consists of two parts (as shown in Fig. 11): 1) Geocast system, which provides location-based situational awareness (LSA) information and 2) OpenStack extension (OSE), which is responsible for deploying, executing, and managing the containers on the edge devices. FocusStack builds a hybrid cloud of edge devices (containers) and DC servers (VMs). When a user initiates a cloud operation through the FocusStack API (such as instantiating a container), the LSA subsystem analyzes the scope of the request based on the Geocast route and sends a resource list of geographic locations to the target area and waits for the response from (online) edge devices that can satisfy the requirements. Then, the selected edge device runs the corresponding OpenStack operations with the help of the conductor module.

The situational-aware subsystem enables one group of devices to monitor the survival status of each other, and each device can update the sensing information of other devices. The geoprocessing projection service is primarily intended to pass requests and reply messages between areas of interest, send out device control information (like drones), and broadcast location-based information. The service is based on a two-layer network, and data can be transmitted over a dedicated WiFi network or the Internet. The sender transmits the data packet to the georouter server that tracks the location and metadata of each edge device, and then, the data packet is sent to each device (including edge devices and a cloud device running a SAMonitor instance) in the specified area based on the address. Location and connection statuses are maintained by the georouting database (GRDB). The GCLib is a software framework that provides data acquisition services, and it runs on edge devices and cloud applications that use SAMonitor. Any application or service in need of the state of the device in the area needs a SAMonitor component to communicate with the LSA subsystem. The application server makes a request to SAMonitor through the FocusStack API, and then, SAMonitor builds a real-time graph of the current area and returns a list of available edge devices. The regional graph is sent to the conductor in the OSE, which is responsible for checking whether the devices are capable of running the tasks, whether the predefined policy rules are met, and so on. Then, the available edge device list is submitted to the application server and the server selects devices to be used. OSE manages and deploys the program through the OpenStack Nova API. The edge devices run a custom version of Nova Compute to interact with the local Docker to manage the container. The container on the edge devices supports all OpenStack services, including access to virtual networks and application-based granularity configuration.

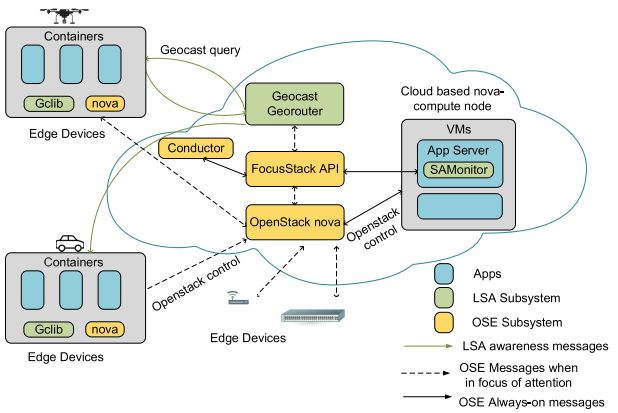


Fig. 11. ***FocusStack system [18].***

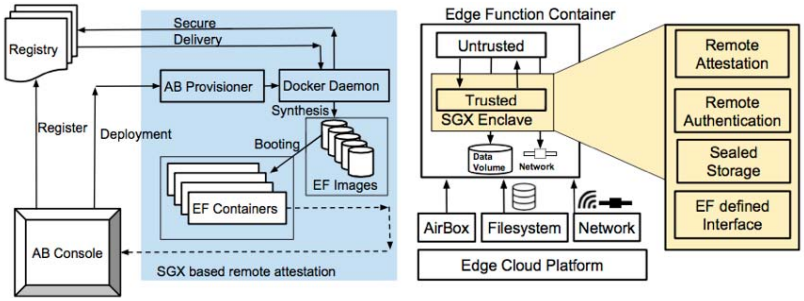


Fig. 12. ***AirBox architecture [19].***

I. AirBox

AirBox [19] is a secure, lightweight, and flexible edge function (EF) system developed by the Georgia Institute of Technology. It supports fast and flexible EF loading and provides service security and user privacy protection. The EF in AirBox is defined as a service that can be loaded on an edge node, and the software stack that supports the EF is named EF Platform (EFP).

As shown in Fig. 12, the AirBox consists of two parts: the AB console and the AB provisioner. The back-end service manager deploys and manages the EF on the edge nodes through the AB console. The AB provisioner that runs at the edge is responsible for providing dynamic EF seamlessly. The EFs are implemented through system-level containers with minimal constraints on developers. Security is enhanced by using the hardware security mechanisms like Intel SGX. AirBox provides centrally controlled backend services in discovering edge nodes and registering them. The AB console is a web-based management system, which activates the docker startup process on the edge node with AB provisioner.

To ensure the security of the AirBox, the EF consists of a trusted part and an untrusted part. The untrusted part is responsible for all network and storage interactions. Based on OpenSGX APIs, AirBox provides four extensible APIs to implement secure communication and storage: remote attestation, remote authentication, sealed storage, and EF-defined interface. AirBox supports a variety of edge features such as aggregation, buffering, and caching.

J. Firework

Wayne State University's MIST Lab proposes a programming model for edge computing—Firework [20]. In the model of Firework, all services/functions are represented in a data view of data sets and functions, which can be the results of processing with their own data or secondary processing with other services/data sets. A system consisting of nodes that apply the Firework model is called the Firework system. In the Firework system, instead of dividing nodes into edge nodes and cloud ones, they are all considered as Firework nodes. Through the mutual invocation of the Firework nodes, the data can be distributed and processed on each node. Along the path of data transmission, the edge nodes perform a series of calculations upon the data, thus forming a "computational flow." The beginning of the computational flow could be user terminals, edge servers close to the user, edge servers close to the cloud, or cloud nodes.

In the Firework system, the data processing service is split into multiple subservices, and the scheduling of the data processing is performed at the following two layers.

1) *Same Subservice Layer Scheduling:* A Firework node can cooperate with another for the same subservices in the surrounding area to achieve optimal response time. For idle Firework nodes, the system can schedule subservice programs onto them, such that a cluster providing a specific subservice is formed dynamically and complete the service faster.
2) *Computational Flow Layer Scheduling:* Firework nodes along the computational flow can cooperate with each other and dynamically schedule execution nodes to achieve an optimal solution. For example, depending on the states of the network, the system can choose Firework nodes for service providing based on the nodes' locations (e.g., selecting those closest to the users).

As shown in Fig. 13, Firework divides the nodes into computing nodes and managers, depending on the type of service those nodes provide. In general, each node with the Firework model has three modules: the job management

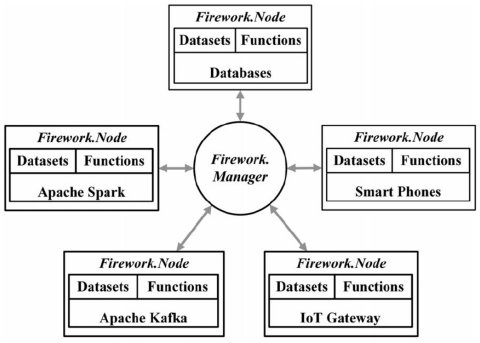


Fig. 13. ***Example of a Firework instance that consists of heterogeneous computing platforms [20].***

Table 1 Summary of Edge Computing Systems

Application Scenarios	Edge Computing Systems	End Devices	Edge Nodes	Computation Architecture	Features/Targets
General Usage Scenario	Cloudlet	Mobile devices	Cloudlet	Hybrid (3-tier)	Lightweight VM migration
	PCloud	Mobile devices	Mobile devices, local server, PC	Hybrid (3-tier)	Resource integration, dynamic allocation
	ParaDrop	IoT devices	Home gateway	Hybrid (3-tier)	Hardware, developer support
	Cachier & Precog	Mobile devices	Mobile devices, local server, PC	Hybrid (3-tier)	Figure recognition, identification
	FocusStack	IoT devices	Router, server	Hybrid (3-tier)	Location-based info, OpenStack extension
	SpanEdge	IoT devices	Local cluster, Cloudlet, Fog	Hybrid (2-tier)	Streaming processing, local/global task
	AirBox	IoT devices	Mobile devices, local server, PC	Hybrid (3-tier)	Security
	CloudPath	Mobile devices	Multi-level data centers	Hybrid (multi-tier)	Path computing
	Firework	Firework.Node	Firework.Node	Two-layer scheduling	Programming model
	Cloud-Sea	Sea	Seaport	Hybrid (3-tier)	Minimal extension, transparency
Vehicular Data Analytics	OpenVDAP	CAVs	XEdge	Hybrid (2-tier)	General platform
	SafeShareRide	Smartphones and vehicles	Smartphones	Hybrid (2-tier)	In-vehicle security
Smart Home	Vigilia	Smart home devices	Hubs	Edge only	Smart Home Security
	HomePad	Smart home devices	Routers	Edge only	Smart Home Security
Video Stream Analytics	LAVEA	~	~	Edge or cloud	Low latency response
	VideoEdge	Cameras	Cameras and private clusters	Hybrid (3-tier)	Resource-accuracy tradeoff
	Video on drones	Autonomous drones	Portable edge computers	Edge only	Bandwidth saving
Virtual Reality	MUVR	Smartphones	Individual households	Edge only	Resource utilization efficiency optimization

module, the actuator management module, and the service management module.

1) *Service Management Module:* This type of module is designed for the management of data views. It provides interfaces to update the data view, as well as relevant programs for data processing.
2) *Job Management Module:* This type of module is responsible for the scheduling, monitoring, and evaluation of the task executions. When the local computing resources are insufficient, the module can look into the node list and the data view and make resource rescheduling at the same subservice layer. When the subservice is running, the module can also provide necessary monitoring information and give feedback to other upstream and downstream nodes for flow layer scheduling.
3) *Actuator Management Module:* This type of module is mainly responsible for managing all hardware resources and hosting the execution processes of different tasks. With the help of this module, the device, running environment and the upper layer functions, could be decoupled, so that the nodes of the Firework system are not limited to a certain type of devices, and the data processing environment is not limited to a certain type of computing platform.

K. Other Edge Computing Systems

The edge systems introduced above depict some typical and basic innovations on the exploitation of edge analytics for highly responsive services. The previous systems are used in general cases, which lay the foundation of further development. We highlight that, in addition to these efforts, there are many other edge systems tuned for a series of different application scenarios. In Table 1, we briefly summarize the general and application-specific edge systems, which leverage different kinds of edge nodes to serve diverse end devices using either a hybrid or edge only computation architecture.

Compared to both on-board computation and cloud-based computation, edge computing can provide more effective data analytics with a lower latency for the moving vehicles [21]. In [21], an open full-stack edge

computing-based platform OpenVDAP is proposed for the data analytics of connected and autonomous vehicles (CAVs). OpenVDAP proposes systematic mechanisms, including varied wireless interfaces, to utilize the heterogeneous computation resources of nearby CAVs, edge nodes, and the cloud. For optimal utilization of the resources, a dynamic scheduling interface is also provided to sense the status of available resources and to offload divided tasks in a distributed way for computation efficiency. SafeShareRide is an edge-based attack detection system addressing in-vehicle security for ridesharing services [22]. Its three detection stages leverage both the smartphones of drivers and passengers as edge computing platform to collect multimedia information in vehicles. Specifically, speech recognition and driving behavior detection stages are first carried out independently to capture in-vehicle danger, and the video capture and uploading stage is activated when abnormal keywords or dangerous behaviors are detected to collect videos for cloud-based analysis. By using such an edge-cloud collaborative architecture, SafeShareRide can accurately detect attacks in-vehicle with low bandwidth demand.

Another scenario that edge computing can play an important role is the IoT devices management in the smart home environment. Wherein, the privacy issue of the wide range of home devices is a popular topic. In [23], the Vigilia system is proposed to harden smart home systems by restricting the network access of devices. A default access deny policy and an API-granularity device access mechanism for applications are adopted to enforce access at the network level. Run time checking implemented in the routers only permits those declared communications, thus helping users secure their home-devices. Similarly, the HomePad system in [24] also proposes to execute IoT applications at the edge and introduces a privacy-aware hub to mitigate security concerns. Homepad allows users to specify privacy policy to regulate how applications access and process their data. Through enforcing applications to use explicit information flow, Homepad can use Prolog rules to verify whether applications have the ability to violate the defined privacy policy at install time.

Edge computing has also been widely used in the analysis of video stream. LAVEA is an edge-based system built for latency-aware video analytics nearby the end users [25]. In order to minimize the response time, LAVEA formulates an optimization problem to determine which part of tasks to be offloaded to the edge computer and uses a task queue prioritizer to minimize the makespan. It also proposes several task placement schemes to enable the collaboration of nearby edge nodes, which can further reduce the overall task completion time. VideoEdge is a system that provides the most promising video analytics implementation across a hierarchy of clusters in the city environment [26]. A three-tier computation architecture is considered with deployed cameras and private clusters as the edge and remote server as the cloud. The hierarchical edge architecture is also adopted in [27] and is believed to be promising in processing live video stream at scale. Technically, VideoEdge searches thousands of combinations of computer vision components implementation, knobs, and placement and finds a configuration to balance the accuracy and resource demands using an efficient heuristic. In [28], a video analytics system for autonomous drones is proposed, where edge computing is introduced to save the bandwidth. Portable edge computers are required here to support dynamic transportation during a mission. Totally, four different video transmission strategies are presented to build an adaptive and efficient computer vision pipeline. In addition to the analytics work (e.g., object recognition), the edge nodes also train filters for the drones to avoid the uploading of the uninteresting video frames.

In order to provide flexible virtual reality (VR) on untethered smartphones, edge computing can be useful to transport the heavy workload from smartphones to their nearby edge cloud [29]. However, the rendering task of the panoramic VR frames (i.e., 2 GB/s) will also saturate the individual households as common edge in the house. In [29], the multiuser virtual reality (MUVR) system is designed to support multiuser VR with efficient bandwidth and computation resources utilization. MUVR is built on a basic observation that the VR frames being rendered and transmitted to different users are highly redundant. For computation efficiency, MUVR maintains a two-level hierarchical cache for invariant background at the edge and the user end to reuse frames whenever necessary. Meanwhile, MUVR transmits a part of all frames in full and delivers the distinct portion for the rest frames to further reduce the transmission costs.

III. OPEN-SOURCE EDGE COMPUTING PROJECTS

Besides the designed edge computing systems for specific purposes, some open-source edge computing projects have also been launched recently. The Linux Foundation published two projects: EdgeX Foundry in 2017 and Akraino Edge Statck [30] in 2018. The Open Network Foundation (ONF) launched a project, namely, CORD [31]. The Apache Software Foundation published Apache Edgent. Microsoft published Azure IoT Edge in 2017 and announced it as an open source in 2018.

Among them, CORD and Akraino Edge Stack focus on providing edge cloud services; EdgeX Foundry and Apache Edgent focus on IoT and aim to solve problems, which bring difficulties to practical applications of edge computing in IoT; Azure IoT Edge provides hybrid cloud-edge analytics, which helps to migrate cloud solutions to IoT devices.

A. CORD

CORD is an open-source project of ONF initiated by AT&T and is designed for network operators. Current network infrastructure is built with closed proprietary-integrated systems provided by network equipment providers. Due to the closed property, the network

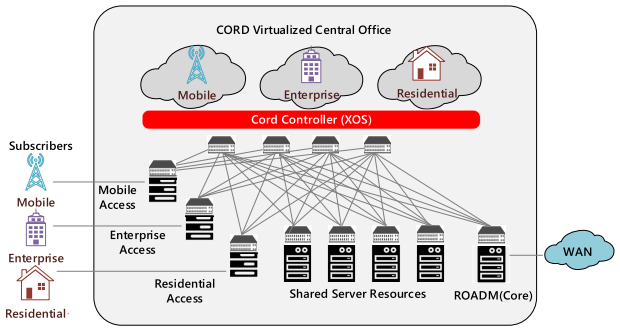


Fig. 14. ***Hardware architecture of CORD.***

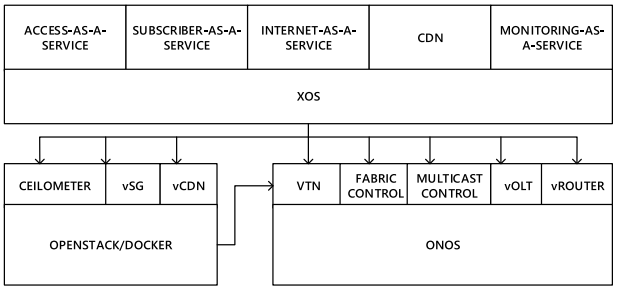


Fig. 15. ***Software architecture of CORD.***

capability cannot scale up and down dynamically. The lack of flexibility results in inefficient utilization of the computing and networking resources. CORD plans to reconstruct the edge network infrastructure to build datacenters with software-defined network (SDN) [32], network function virtualization (NFV) [33], and Cloud technologies. It attempts to slice the computing, storage, and network resources so that these datacenters can act as clouds at the edge, providing agile services for end users.

CORD is an integrated system built from commodity hardware and open-source software. Fig. 14 shows the hardware architecture of CORD [31]. It uses commodity servers that are interconnected by a Fabric of White-box switches. White-box switch [34] is a component of SDN switch, which is responsible to regulate the flow of data according to SDN controller. These commodity servers provide computing, storage resources, and the fabric of switches are used to build the network. This switching fabric is organized to a spine-leaf topology [35], a kind of flat network topology structure, which adds a horizontal network structure parallel to the trunk longitudinal network structure and then adds corresponding switching network on the horizontal structure. Comparing to the traditional three-tier network topology, it can provide scalable throughput for greater East-to-West network traffic, that is, traffic coming from network diagram drawings that usually depict local area network (LAN) traffic horizontally. In addition, specialized access hardware is required to connect subscribers. The subscribers can be divided into three categories for different use cases, mobile subscribers, enterprise subscribers, and residential subscribers. Each category demands different access hardware due to different access technologies. In terms of software, Fig. 15 shows the software architecture of CORD [31]. Based on the servers and the fabric of switches, OpenStack provides with IaaS capability for CORD, it manages the compute, storage, and networking resources as well as creating VMs and virtual networks. Docker is used to run services in containers for isolation. Open Network Operating System (ONOS) is a network operating system that is used to manage network components like the switching fabric and provide communication services to end-users. XOS provides a control plane to assemble and compose services. Other software projects provide component capabilities, for example, vRouter (Virtual Router) provides virtual routing functionality.

The edge of the operator network is a sweet spot for edge computing because it connects customers with operators and is close to customers' applications as data sources. CORD takes edge computing into consideration and moves to support edge computing as a platform to provide edge cloud services (from the released version 4.1). CORD can be deployed into three solutions: mobile CORD (M-CORD), residential CORD (R-CORD), and enterprise CORD (E-CORD) for different use cases. M-CORD focuses on the mobile network, especially 5G network, and it plans to disaggregate and virtualize cellular network functions to enable services to be created and scaled dynamically. This agility helps to provide multiaccess edge services for mobile applications. For those use cases like driverless cars or drones, users can rent the edge service to run their edge applications. Similarly, R-CORD and E-CORD are designed to be agile service delivery platforms but for different users, residential, and enterprise users, relatively.

So far, the deployment of CORD is still under test among network operators, and more research is needed to combine CORD with various edge applications.

B. Akraino Edge Stack

Akraino Edge Stack, initiated by AT&T and now hosted by Linux Foundation, is a project to develop a holistic solution for edge infrastructure so as to support high-availability edge cloud services [30]. An open-source software stack, as the software part of this solution, is developed for network carrier to facilitate optimal networking and workload orchestration for underlying infrastructure in order to meet the need of edge computing such as low latency, high performance, high availability, scalability, and so on.

To provide a holistic solution, Akraino Edge Stack has a wide scope from the infrastructure layer to the application layer. Fig. 16 [30] shows the scope with three layers. In the application layer, Akraino Edge Stack wants to create a virtual network function (VNF) ecosystem and calls for edge applications. The second layer consists of middleware that supports applications in the top layer.

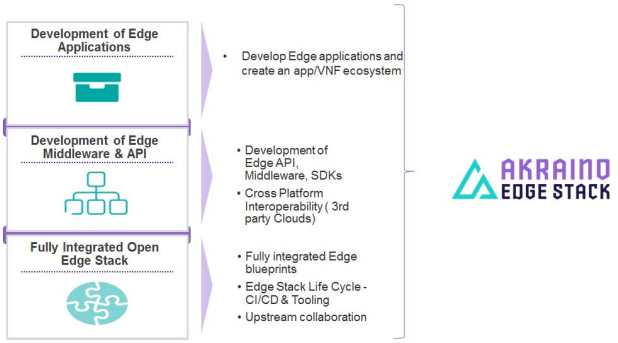


Fig. 16. ***Akraino Edge Stack's scope.***

In this layer, Akraino Edge Stack plans to develop Edge API and framework for interoperability with third-party Edge projects such as EdgeX Foundry. At the bottom layer, Akraino Edge Stack intends to develop an open-source software stack for the edge infrastructure in collaboration with upstream communities. It interfaces with and maximizes the use of existing open-source projects such as Kubernetes, OpenStack, and so on. Akraino Edge Stack provides different edge use cases with blueprints, which are declarative configurations of the entire stack including hardware, software, point of delivery, and so on [30]. The application domains of these blueprints start from Telco industry and are expected to be applied in more domains like enterprise and industrial IoT. Now, Akraino Edge Stack has put forward several blueprints such as micro-MEC and edge media processing. Micro-MEC intends to develop a new service infrastructure for smart cities, which enables developing services for smart city and has high data capacity for citizens. Edge media processing intends to develop a network cloud to enable real-time media processing and edge media AI analytics with low latency.

As an emerging project, Akraino Edge Stack has been taken to execution since August 2018. Thus, more research needs to be done with the development of this project.

C. EdgeX Foundry

EdgeX Foundry is a standardized interoperability framework for IoT edge computing, whose sweet spots are edge nodes such as gateways, hubs, and routers [36]. It can connect with various sensors and devices via different protocols, manage them and collect data from them, and export the data to a local application at the edge or the cloud for further processing. EdgeX is designed to be agnostic to hardware, CPU, operating system, and application environment. It can run natively or run in Docker containers.

Fig. 17 [36] shows the architecture of EdgeX Foundry. "South side" at the bottom of the figure includes all IoT objects, and the edge of the network that communicates directly with those devices, sensors, actuators, and other IoT objects to collect the data from them. Relatively, "north side" at the top of the figure includes the cloud (or enterprise system) where data are collected, stored, aggregated, analyzed, and turned into information, and the part of the network that communicates with the Cloud. EdgeX Foundry connects these two sides regardless of the differences in hardware, software, and network. EdgeX tries to unify the manipulation method of the IoT objects from the south side to a common API so that those objects can be manipulated in the same way by the applications of the north side.

EdgeX uses a device profile to describe a south side object. A device profile defines the type of the object, the format of data that the object provides, the format of data to be stored in EdgeX, and the commands used to manipulate this object. Each device profile involves a device service, which is a service that converts the format of the data and translates the commands into instructions that IoT objects know how to execute. EdgeX provides SDK for developers to create device services so that it can support any combination of device interfaces and protocols by programming.

EdgeX consists of a collection of microservices, which allows services to scale up and down based on device capability. These microservices can be grouped into four service layers and two underlying augmenting system services, as depicted in Fig. 17. The four service layers include device services layer, core services layer, supporting services layer, and export services layer, respectively; the two underlying augmenting system services are system management and security, respectively. Each of the six layers consists of several components and all components use a common restful API for configuration.

1) *Device Services Layer:* This layer consists of device services. According to the device profiles, device service layer converts the format of the data, sends them to core services layer, and translates the command requests from the core services layer.
2) *Core Services Layer:* This layer consists of four components: core data, command, metadata, and registry and configuration. Core data is a persistence repository as well as a management service. It stores and manages the data collected from the south side objects. Command is a service to offer the API for command requests from the north side to device services. Metadata is a repository and management service for metadata about IoT objects. For example, the device profiles are uploaded and stored in metadata. Registry and configuration provide a centralized management of configuration and operating parameters for other microservices.
3) *Supporting Services Layer:* This layer is designed to provide edge analytics and intelligence [33]. Now, the rules engine, alerting and notification, scheduling, and logging microservices are implemented. A target range of data can be set to trigger a specific device actuation as a rule and rules engine helps to

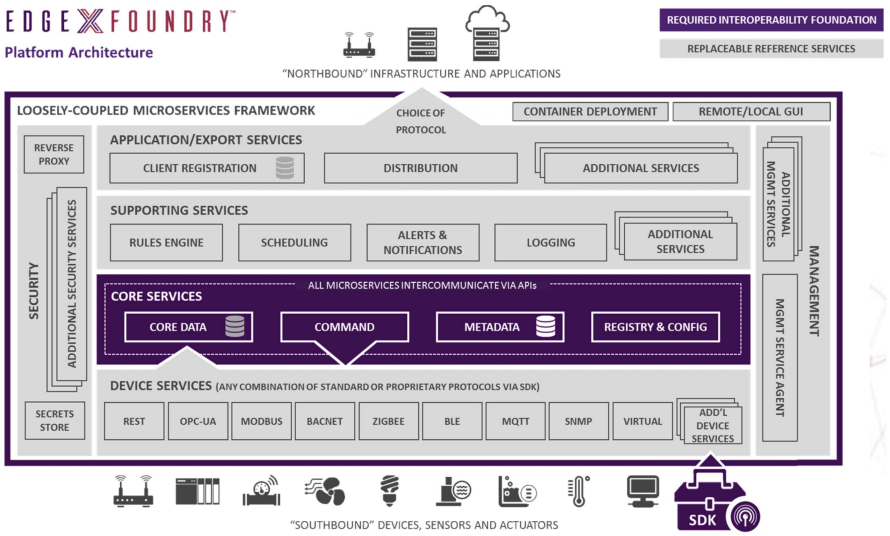


Fig. 17. ***Architecture of EdgeX Foundry [36].***

realize the rule by monitoring the incoming data. Alerting and notifications can send notifications or alerts to another system or person by email, REST callback, or other methods when an urgent actuation or a service malfunction happens. The scheduling module can set up a timer to regularly clean up the stale data. Logging is used to record the running information of EdgeX.

4) *Export Services Layer:* This layer connects EdgeX with north side and consists of client registration and export distribution. Client registration enables clients like a specific cloud or a local application to register as recipients of data from core data. Export distribution distributes the data to the clients registered in client registration.
5) *System Management and Security:* System management provides management operations, including installation, upgrade, starting, stopping, and monitoring, as EdgeX is scalable and can be deployed dynamically. Security is designed to protect the data and command of IoT objects connected with EdgeX Foundry.

EdgeX is designed for the user cases dealing with multitudes of sensors or devices, such as automated factories, machinery systems, and a lot of other cases in IoT. Now, EdgeX Foundry is in the rapid upgrading phase, and more features will be added in future releases. An EdgeX UI is in development as a web-based interface to add and manage the devices.

D. Apache Edgent

Apache Edgent, which was known as Apache Quarks previously, is an Apache Incubator project at present. It is an open-source programming model for lightweight runtime data analytics, used in small devices such as routers and gateways at the edge. Apache Edgent focuses on data analytics at the edge, aiming to accelerate the development of data analysis.

As a programming model, Edgent provides API to build edge applications. Fig. 18 illustrates the model of the Edgent applications. Edgent uses a topology as a graph to represent the processing transformation of streams of data, which are abstracted to a Tstream class. A connector is used to get streams of data from external entities such as sensors and devices in physical world or to send streams of data to back-end systems like a cloud. The primary API of Edgent is responsible for data analysis. The streams of data can be filtered, split, transformed, or processed by other operations in a topology. Edgent uses a provider to act as a factory to create and execute topologies. To build an Edgent application, users should first get a provider, then create a topology and add the processing flow to deal with the streams of data, and finally, submit the topology. The deployment environments of Edgent are Java 8, Java 7, and Android.

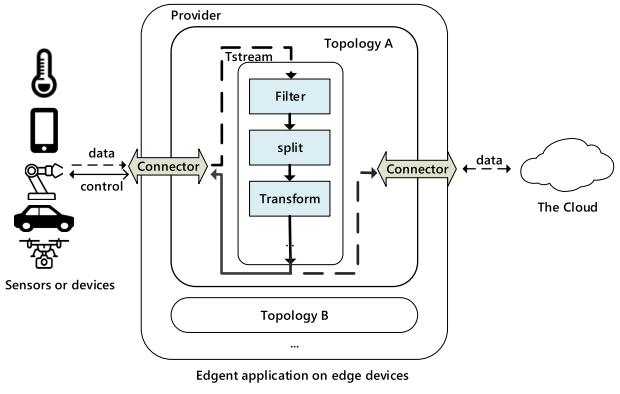


Fig. 18. ***Model of the Edgent applications.***

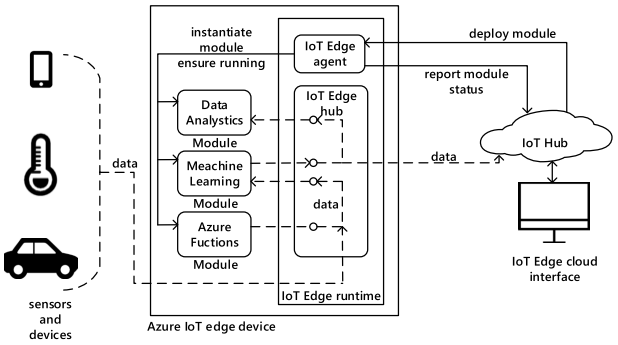


Fig. 19. ***Diagram of Azure IoT Edge.***

Edgent provides APIs for sending data to back-end systems and now supports message queuing telemetry transport (MQTT), IBM Watson IoT Platform, Apache Kafka, and custom message hubs. Edgent applications analyze the data from sensors and devices and send the essential data to the back-end system for further analysis. For IoT use cases, Edgent helps to reduce the cost of transmitting data and provide local feedback.

Edgent is suitable for use cases in IoT such as intelligent transportation, automated factories, and so on. In addition, the data in Edgent applications are not limited to sensor readings, and they can also be files or logs. Therefore, Edgent can be applied to other use cases. For example, it can perform a local data analysis when embedded in application servers, where it can analyze error logs without impacting network traffic [37].

E. Azure IoT Edge

Azure IoT Edge, provided by Microsoft Azure as a cloud service provider, tries to move cloud analytics to edge devices. These edge devices can be routers, gateways, or other devices, which can provide computing resources. The programming model of Azure IoT Edge is the same as that of other Azure IoT services [38] in the cloud, which enables the user to move their existing applications from Azure to the edge devices for lower latency. The convenience simplifies the development of edge applications. In addition, Azure services such as Azure functions, Azure ML, and Azure stream analytics can be used to deploy complex tasks on the edge devices such as ML, image recognition, and other tasks about AI.

Azure IoT Edge consists of three components: IoT Edge modules, IoT Edge runtime, and a cloud-based interface, as depicted in Fig. 19. The first two components run on edge devices, and the last one is an interface in the cloud. IoT Edge modules are containerized instances running the customer code or Azure services. IoT Edge runtime manages these modules. The cloud-based interface is used to monitor and manage the former two components, in other words, monitor and manage the edge devices.

IoT Edge modules are the places that run specific applications as the units of execution. A module image is a docker image containing the user code. A module instance, as a Docker container, is a unit of computation running the module image. If the resources at the edge devices are sufficient, these modules can run the same Azure services or custom applications as in the cloud because of the same programming model. In addition, these modules can be deployed dynamically as Azure IoT Edge is scalable.

IoT Edge runtime acts as a manager on the edge devices. It consists of two modules: IoT Edge hub and IoT Edge agent. IoT Edge hub acts as a local proxy for IoT Hub, which is a managed service, and a central message hub in the cloud. As a message broker, IoT Edge hub helps modules to communicate with each other and transport data to IoT Hub. IoT Edge agent is used to deploy and monitor the IoT Edge modules. It receives the deployment information about modules from IoT Hub, instantiates these modules, and ensures they are running, for example, restarts the crashed modules. In addition, it reports the status of the modules to the IoT hub.

IoT Edge cloud interface is provided for device management. By this interface, users can create edge applications, then send these applications to the device, and finally, monitor the running status of the device. This monitoring function is useful for use cases with massive devices, where users can deploy applications to devices on a large scale and monitor these devices.

A simple deployment procedure for applications is that: users choose an Azure service or write their own code as an application, build it as an IoT Edge module image, and deploy this module image to the edge device with the help of the IoT Edge interface. Then, the IoT Edge receives the deployment information, pulls the module image, and instantiates the module instance.

Azure IoT Edge has wide application areas. Now, it has application cases on intelligent manufacturing, irrigation system, drone management system, and so on. It is worth noting that Azure IoT Edge is open-source but the Azure services such as Azure functions, Azure ML, and Azure stream are charged.

F. Comparative Study

We summarize the features of the above-mentioned open-source edge computing systems in Table 2. Then, we compare them from different aspects in Table 2, including the main purpose of the systems, application area, deployment, target user, virtualization technology, system characteristic, limitations, scalability, and mobility. We believe that such comparisons give better understandings of the current open-source edge computing systems.

1) Main Purpose: The main purpose shows the target problem that a system tries to fix, and it is a key factor for us to choose a suitable system to run edge applications. As an interoperability framework, EdgeX Foundry aims to communicate with any sensor or device in IoT. This ability is necessary for edge applications with data from various sensors and devices. Azure IoT Edge offers an

Table 2 Comparison of Open Edge System Characteristics

Aspect	EdgeX Foundry	Azure IoT Edge	Apache Edgent	CORD	Akraino Edge Stack
User access interface	Restful API or EdgeX UI	Web service, Command-line	API	API or XOS-GUI	N/A
OS support	Various OS	Various OS	Various OS	Ubuntu	Linux
Programming framework	Not provided	Java, .NET, C, Python, etc.	Java	Shell script, Python	N/A
Main purpose	Provide with Interoperability for IoT edge	Support hybrid cloud-edge analytics	Accelerate the development process of data analysis	Transform edge of the operator network into agile service delivery platforms	Support edge clouds with an open source software stack
Application area	IoT	Unrestricted	IoT	Unrestricted	Unrestricted
Deployment	Dynamic	Dynamic	Static	Dynamic	Dynamic
Target user	General users	General users	General users	Network operators	Network operators
Virtualization technology	Container	Container	JVM	Virtual Machine and Container	Virtual Machine and Container
System characteristics	A common API for device management	Powerful Azure services	APIs for data analytics	Widespread edge clouds	Widespread edge clouds
Limitation	Lack of programable interface	Azure Services is chargeable	Limited to data analytics	Unable to be offline	Unable to be offline
Scalability	Scalable	Scalable	Not scalable	Scalable	Scalable
Mobility	Not support	Not support	Not support	Support	Support

efficient solution to move the existing applications from cloud to edge and to develop edge applications in the same way with the cloud applications. Apache Edgent helps to accelerate the development process of data analysis in IoT use cases. CORD aims to reconstruct the current edge network infrastructure to build datacenters so as to provide agile network services for end-user customers. From the view of edge computing, CORD provides with multiaccess edge services. Akraino edge stack provides an open-source software stack to support high-availability edge clouds.

2) Application Area: EdgeX Foundry and Apache Edgent both focus on IoT edge, and EdgeX Foundry is geared toward communication with various sensors and devices, while Edgent is geared toward data analysis. They are suitable for intelligent manufacturing, intelligent transportation, and smart city where various sensors and devices generate data all the time. Azure IoT Edge can be thought as the expansion of Azure Cloud. It has an extensive application area but depends on the computation resources of edge devices. In addition, it is very convenient to deploy edge applications about AI such as ML and image recognition to Azure IoT Edge with the help of Azure services. CORD and Akraino Edge Stack support edge cloud services, which have no restriction on the application area. If the edge devices of users do not have sufficient computing capability, these two systems are suitable for users to run resource-intensive and interactive applications in connection with the operator network.

3) Deployment: As for the deployment requirements, EdgeX Foundry, Apache Edgent, and Azure IoT Edge are deployed in edge devices such as routers, gateways, switchers, and so on. Users can deploy EdgeX Foundry by themselves, add or reduce microservices dynamically, and run their own edge applications. Differently, users need the help of cloud-based interface to deploy Azure IoT Edge and develop their edge applications. CORD and Akraino Edge Stack are designed for network operators, who need fabric switches, access devices, network cards, and other related hardware apart from compute machines. Customers have no need to think about the hardware requirements and management process of the hardware, but to rent the services provided by the network operators like renting a cloud service instead of managing a physical server.

4) Target User: Although these open-source systems focus on edge computing, their target users are not the same. EdgeX Foundry, Azure IoT Edge, and Apache Edgent have no restriction on target users. Therefore, every developer can deploy them into local edge devices such as gateways, routers, and hubs. Differently, CORD and Akraino Edge Stack are created for network operators because they focus on edge infrastructure.

5) Virtualization Technology: At present, virtualization technologies are widely used. VM technology can provide better management and higher utilization of resources, stability, scalability, and other advantages. Container technology can provide services with isolation and agility but with negligible overhead, which can be used in edge devices [39]. Using OpenStack and Docker as software components, CORD and Akraino Edge Stack use both of these two technologies to support edge cloud. Different edge devices may have different hardware and software environments. For those edge systems that are deployed on edge devices, a container is a good technology for services to keep independence in different environments. Therefore, EdgeX Foundry and Azure IoT Edge choose to run as Docker containers. As for Edgent, Edgent applications run on Java virtual machine (JVM).

6) System Characteristic: System characteristics show the unique features of the system, which may help users to develop, deploy, or monitor their edge applications. It will

save a lot of workload and time if making good use of these characteristics. EdgeX Foundry provides a common API to manage the devices, and this brings great convenience to deploying and monitoring edge applications in a large scale. Azure IoT Edge provides powerful Azure services to accelerate the development of edge applications. Apache Edgent provides a series of functional APIs for data analytics, which lowers the difficulty and reduces the time for developing edge analytic applications. CORD and Akraino Edge Stack provide with multiaccess edge services on edge cloud. We only need to keep a connection with the operator network, and we can apply for these services without the need to deploy an edge computing system on edge devices by ourselves.

7) Limitation: This section discusses the limitation of the latest version of them to deploy edge applications. The latest version of EdgeX Foundry has not provided a programmable interface in its architecture for developers to write their own applications. Although EdgeX allows us to add custom implementations, it demands more workload and time. As for Azure IoT Edge, although it is open-source and free, Azure services are chargeable as commercial software. For Apache Edgent, it is lightweight and it focuses on only data analytics. As for CORD and Akraino Edge Stack, these two systems demand a stable network between data sources and the operators because the edge applications are running on the edge of the operator network rather than local devices.

8) Scalability: Increasing applications at edge make the network architecture more complex and the application management more difficult. Scalability is one major concern in edge computing. Among these edge computing systems, Azure IoT Edge, CORD, and Akraino Edge Stack apply Docker technology or VM technology to support users to scale up or down their applications efficiently by adding or deleting module images. EdgeX Foundry is also a scalable platform that enables users to dynamically add or reduce microservices to adapt to the actual needs. However, Apache Edgent is not scalable enough because every Edgent application is a single Java application and performance cannot be changed dynamically.

9) Mobility: For EdgeX Foundry, Apache Edgent, and Azure IoT Edge, once the applications are executed on some edge devices, they cannot be dynamically migrated to other devices. CORD and Akraino Edge Stack, deployed in the telecom infrastructure, support mobile edge services through mobile access network like 4G/5G. The mobility of these systems meets the need for cases such as unmanned cars and drones.

10) Scenarios: We discuss the choice of open-source tools from the perspective of the following three scenarios, as shown in Fig. 20.

In the first scenario, suppose IoT edge applications are running on LAN, and the local enterprise system is regarded as a back-end system with no need for third-party clouds. In this case, EdgeX Foundry or Apache Edgent is favorable because they enable users to build and control their own back-end system without being bound to any specific cloud platform. Furthermore, for the sake of managing and controlling edge devices on a large scale, EdgeX Foundry is a better choice for good device management capability. If considering the data analysis, Apache Edgent is preferable to EdgeX Foundry. It provides a programming model to accelerate the development of edge analytic applications and a set of powerful APIs for data processing.

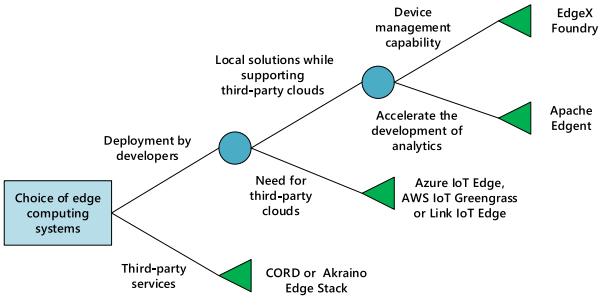


Fig. 20. ***Choice of open-source tools in different scenarios.***

In the second scenario, suppose the cloud services push to the edge of the network to improve the quality. In this case, Auzre IoT Edge provides a convenient way for developers to migrate the applications from the cloud to the edge devices and to leverage third-party high-value services or functions when developing edge systems. In addition, AWS IoT Greengrass and Link IoT Edge, which are published by Amazon and Alibaba Cloud, are good choices as the competitive projects. More specifically, Azure IoT Edge provides Azure services such as Azure functions, Azure stream analytics, and Azure ML. AWS IoT Greengrass can run AWS Lamda functions and ML models that are created, trained, and optimized in the cloud. Link IoT Edge provides function compute and other functions. Based on the application requirements, a suitable system could be chosen among them by taking account the functions they provide.

In the third scenario, suppose the users expect to directly leverage third-party services to deploy edge applications without any hardware or software system locally. In this case, edge systems such as CORD and Akraino Edge Stack are suitable. The users could choose one of them dependent on their application requirements. These systems are deployed by telecom operators. Telecom operators are also the providers of network services. Thus, when there exist special network requirements of edge applications, these systems could satisfy the requirements. For example, edge computing applications on unmanned vehicles or drones need the support of wireless telecom networks (such as 4G/5G); in this case, a MEC service provided by these systems is a good choice.

In addition to the systems described above, there are other emerging open-source projects. Device management edge, as part of Mbed IoT Platform published by the Advanced RISC Machine (ARM), is responsible for edge

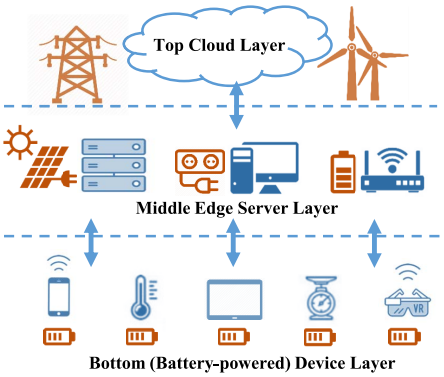


Fig. 21. ***Three-layer edge computing paradigm from a power source view.***

computing and provides the ability to access, manage, and control edge devices. KubeEdge, released by Huawei, provides edge nodes with native containerized application orchestration capabilities.

IV. ENHANCING ENERGY EFFICIENCY OF EDGE COMPUTING SYSTEMS

In the design of an edge computing system, energy consumption is always considered as one of the major concerns and evaluated as one of the key performance metrics. In this section, we review the energy-efficiency-enhancing mechanisms adopted by the state-of-the-art edge computing systems, from the view of the top cloud layer, middle edge server layer, and bottom device layer, respectively (the three-layer paradigm is illustrated by Fig. 21).

A. At the Top Cloud Layer

For cloud computing, a centralized dc can comprise thousands of servers and thus consume enormous energy [40]. As an alternative to the cloud computing, does the edge/fog computing paradigm consume more or less energy? Different points of view have been given: some claim that decentralized data storage and processing supported by the edge computing architecture are more energy efficient [41], [42], while some others show that such distributed content delivery may consume more energy than that of the centralized way [43].

Jalali *et al.* [44] give a thorough energy analysis for applications running over the centralized DC (i.e., under cloud mode) and decentralized nano DCs (i.e., under fog mode), respectively. The results indicate that the fog mode may be with a higher energy efficiency, depending on several system design factors (e.g., type of application, type of access network, and ratio of active time), and those applications that generate and distribute a large amount of data in end-user premises result in the best energy saving under the fog mode.

B. At the Middle Edge Server Layer

At the middle layer of the edge computing paradigm, energy is also regarded as an important aspect, as the edge servers can be deployed in a domestic environment or powered by the battery (e.g., a desktop or a portable WiFi router, as shown in Fig. 21). Thus, to provide a higher availability, many power management techniques have been applied to limit the energy consumption of edge servers while still ensuring their performances. We give a review of two major strategies used at the edge server layer in recent edge computing systems.

1) Low-Power System Design and Power Management: In [45], the tactical cloudlet is presented and its energy consumption when performing VM synthesis is evaluated particularly, under different cloudlet provisioning mechanisms. The results show that the largest amount of energy is consumed by: 1) VM synthesis due to the large payload size and 2) on-demand VM provisioning due to the long application-ready time. Such results lead to the high energy efficiency policy: combining cached VM with cloudlet push for cloudlet provision.

A service-oriented architecture for fog/edge computing, Fog Data, is proposed and evaluated in [46]. It is implemented with an embedded computer system and performs data mining and data analytics on the raw data collection from the wearable sensors (in telehealth applications). With Fog Data, orders of magnitude data are reduced for transmission, thus leading to enormous energy saving. Furthermore, Fog Data is with a low power architecture design and even consumes much less energy than that of a Raspberry Pi.

In [47], a performance-aware orchestrator for Docker containers, named DockerCap, is developed to meet the power consumption constraints of the edge server (fog node). Following the observe-decide-act loop structure, DockerCap is able to manage container resources at run-time and provide soft-level power capping strategies. The experiments demonstrate that the obtained results with DockerCap are comparable to that from the power capping solution provided by the hardware (Intel RAPL).

An energy-aware edge computer architecture is designed to be portable and usable in the fieldwork scenarios in [48]. Based on the architecture, a high-density cluster prototype is built using the compact general-purpose commodity hardware. Power management policies are implemented in the prototype to enable the real-time energy awareness. Through various experiments, it shows that both the load balance strategies and cluster configurations have big impacts on the system energy consumption and responsiveness.

2) Green-Energy-Powered Sustainable Computing: Dual-energy sources are employed to support the running of a fog computing-based system in [49], where solar power is utilized as the primary energy supply of the fog nodes. A comprehensive analytic framework is presented to minimize the long-term cost of energy consumption. Meanwhile, the framework also enables an energy-efficient data offloading (from fog nodes to the cloud) mechanism to help provide a high quality of service.

In [50], a rack-scale green-energy-powered edge infrastructure, *in situ* server system using renewable energy (InSURE) is implemented for data preprocessing at the edge. InSURE can be powered by standalone (solar/wind) power and with batteries as the energy backup. Meanwhile, an energy buffering mechanism and a joint spatiotemporal power management scheme are applied to enable efficient energy flow control from the power supply to the edge server.

C. At the Bottom Device Layer

As a well-recognized fact, the IoT devices in edge computing usually have strict energy constraints, e.g., limited battery life and energy storage. Thus, it remains a key challenge to power a great number (can up to tens of billions) of IoT devices at the edge, especially for those resource-intensive applications or services [51]. We review the energy-saving strategies adopted at the device layer of the edge computing diagram. Specifically, we go through three major approaches to achieving high energy efficiency in different edge/fog computing systems.

1) Computation Offloading to Edge Servers or Cloud: As a natural idea to solve the energy poverty problem, computation offloading from the IoT devices to the edge servers or cloud has been long investigated [52]–[54]. It was also demonstrated that, for some particular applications or services, offloading tasks from IoT devices to more powerful ends can reduce the total energy consumption of the system since the task execution time on powerful servers or cloud can be much shortened [55]. Although it increases the energy consumption of (wireless) data transmission, the tradeoff favors the offloading option as the computational demand increases [56].

Having realized that the battery life is the primary bottleneck of handheld mobile devices, Cuervo *et al.* [54] present the mobile assistance using infrastructure (MAUI), an architecture for mobile code offloading and remote execution. To reduce the energy consumption of the smartphone program, MAUI adopts a fine-grained program partitioning mechanism and minimizes the code changes required at the remote server or cloud. The ability of MAUI in energy reduction is validated by various experiments upon macrobenchmarks and microbenchmarks. The results show that MAUI's energy saving for a resource-intensive mobile application is up to one order of magnitude, also with a significant performance improvement.

Like MAUI [54], Chun *et al.* [56] design and implement CloneCloud, a system that helps partition mobile application programs and performs strategically offloading for fast and elastic execution at the cloud end. As the major difference to MAUI, CloneCloud involves less programmer help during the whole process and only offloads particular program partitions on demand of execution, which further speeds up the program execution. Evaluation shows that CloneCloud can improve the energy efficiency of mobile applications (along with their execution efficiency) by 20 times. Similarly, in [57], by continuous updates of software clones in the cloud with a reasonable overhead, the offloading service can lead to energy reduction at the mobile end by a factor. For computation-intensive applications on resource-constrained edge devices, their executions usually need to be offloaded to the cloud. To reduce the response latency of the image recognition application, Precog is presented, which has been introduced in Section II-G. With the on-device recognition caches, Precog much reduces the amount of images offloading to the edge server or cloud, by predicting and prefetching the future images to be recognized.

2) Collaborated Devices Control and Resource Management: For energy saving of the massive devices at the edge, besides offloading their computational tasks to more powerful ends, there is also a great potential via sophisticated collaboration and cooperation among the devices themselves. Particularly, when the remote resources from the edge server or cloud are unavailable, it is critical and nontrivial to complete the edge tasks while without violating the energy constraint.

PCloud is presented in [10] to enhance the capability of individual mobile devices at the edge. By seamless using available resources from the nearby devices, PCloud forms a personal cloud to serve end users whenever the cloud resources are difficult to access, where device participation is guided in a privacy-preserving manner. The authors show that, by leveraging multiple nearby device resources, PCloud can much reduce the task execution time as well as energy consumption. For example, in the case study of neighborhood watch with face recognition, the results show a 74% reduction in energy consumption on a PCloud versus on a single-edge device. Similar to PCloud, the concept of mobile device cloud (MDC) is proposed in [58], where computational offloading is also adopted among the mobile devices. It shows that the energy efficiency (gain) is increased by 26% via offloading in MDC. Liu *et al.* [59] propose an adaptive method to dynamically discovery available nearby resource in heterogeneous networks and perform automatic transformation between centralized and flooding strategies to save energy.

As current long-term evolution (LTE) standard is not optimized to support a large simultaneous access of IoT devices, Abdelwahab *et al.* [60] propose an improved memory replication architecture and protocol, REPLISON, for computation offloading of the massive IoT devices at the edge. REPLISON improves the memory replication performance through an LTE-optimized protocol, where device-to-device (D2D) communication is applied as an important supporting technology (to pull memory replicas from IoT devices). The total energy consumption of REPLISON is generally worse than the conventional LTE scenario as it needs more active devices. However, the energy consumed per device during a single replicate transmission is much less. With further evaluation results, it shows that

Table 3 Comparison of Deep learning Systems on Edge

Features	AWS IoT Greengrass	Azure IoT Edge	Cloud IoT Edge
Developer	Amazon	Microsoft	Google
Components	IoT Greengrass Core, IoT Device SDK, IoT Greengrass SDK	IoT Edge modules, IoT Edge runtime, Cloud-based interface	Edge Connect, Edge ML, Edge TPU
OS	Linux, macOS, Windows	Windows, Linux, macOS	Linux, macOS, Windows, Android
Target device	Multiple platforms (GPU-based, Raspberry Pi)	Multiple platforms	TPU
Characteristic	Flexible	Windows friendly	Real-time

REPLISOM has an energy advantage over the conventional LTE scenarios as long as the size of replica is sufficiently small.

For IoT devices distributed at the edge, Abdelwahab *et al.* [61] leverage the software agents running on the IoT devices to establish an integrated multiagent system (MAS). By sharing data and information among the mobile agents, edge devices are able to collaborate with each other and improve the system energy efficiency in executing distributed opportunistic applications. Upon the experimental platform with 100 sensor nodes and 20 smartphones as edge devices, the authors show the great potential of data transmission reduction with MAS. This leads to a significant energy saving, from 15% to 66%, under different edge computing scenarios. As another work applying data reduction for energy saving, CAROMM [62] employs a change detect technique (LWC algorithm) to control the data transmission of IoT devices while maintaining the data accuracy.

V. DEEP LEARNING OPTIMIZATION AT THE EDGE

In the past decades, we have witnessed the burgeoning of ML, especially deep learning-based applications that have changed human being's life. With complex structures of hierarchical layers to capture features from raw data, deep learning models have shown outstanding performances in those novel applications, such as machine translation, object detection, and smart question and answer systems.

Traditionally, most deep learning based applications are deployed on a remote cloud center, and many systems and tools are designed to run deep learning models efficiently on the cloud. Recently, with the rapid development of edge computing, the deep learning functions are being offloaded to the edge. Thus, it calls for new techniques to support the deep learning models at the edge. This section classifies these technologies into three categories: systems and toolkits, deep learning packages, and hardware.

A. Systems and Toolkits

Building systems to support deep learning at the edge is currently a hot topic for both industry and academy. There are several challenges when offloading state-of-the-art AI techniques on the edge directly, including computing power limitation, data sharing and collaborating, and mismatch between edge platform and AI algorithms. To address these challenges, OpenEI is proposed as an Open Framework for Edge Intelligence [63]. OpenEI is a lightweight software platform to equip edges with intelligent processing and data sharing capability. OpenEI consists of three components: a package manager to execute the real-time deep learning task and train the model locally, a model selector to select the most suitable model for different edge hardware, and a library including a RESTFul API for data sharing. The goal of OpenEI is that any edge hardware will have intelligent capability after deploying it.

In the industry, some top-leading tech-giants have published several projects to move the deep learning functions from the cloud to the edge. Except Microsoft published Azure IoT Edge that has been introduced in Section III-E, Amazon and Google also build their services to support deep learning on the edge. Table 3 summarizes the features of the systems, which will be discussed in the following.

AWS has published IoT Greengrass ML Inference [64] after IoT Greengrass. AWS IoT Greengrass ML Inference is a software to support ML inferences on local devices. With AWS IoT Greengrass ML Inference, connected IoT devices can run AWS Lambda functions and have the flexibility to execute predictions based on those deep learning models created, trained, and optimized in the cloud. AWS IoT Greengrass consists of three software distributions: AWS IoT Greengrass Core, AWS IoT Device SDK, and AWS IoT Greengrass SDK. Greengrass is flexible for users as it includes a prebuilt TensorFlow, Apache MXNet, and Chainer package, and it can also work with Caffe2 and Microsoft Cognitive Toolkit.

Cloud IoT Edge [65] extends Google Cloud's data processing and ML to edge devices by taking advantages of Google AI products, such as TensorFlow Lite and Edge tensor processing unit (TPU). Cloud IoT Edge can either run on Android or Linux-based operating systems. It is made up of three components: edge connect ensures the connection to the cloud and the updates of software and firmware, edge ML runs ML inference by TensorFlow Lite, and edge TPU specific designed to run TensorFlow Lite ML models. Cloud IoT Edge can satisfy the real-time requirement for the mission-critical IoT applications as it can take advantages of Google AI products (such as TensorFlow Lite and Edge TPU) and optimize the performance collaboratively.

B. Deep Learning Packages

Many deep learning packages have been widely used to deliver the deep learning algorithms and deployed on the cloud DCs, including TensorFlow [66], Caffe [67], PyTorch [68], and MXNet [69]. Due to the limitations of computing resources at the edge, the packages designed

Table 4 Comparison of Deep Learning Packages on Edge

Features	TensorFlow Lite	Caffe2	PyTorch	MXNet	CoreML	TensorRT
Developer	Google	Facebook	Facebook	DMLC, Amazon	Apple	NVIDIA
Open Source License	Apache-2.0	Apache-2.0	BSD	Apache-2.0	Not open source	Not open source
Task	Inference	Training, Inference	Training, Inference	Training, Inference	Inference	Inference
Target Device	Mobile and embedded device	Multiple platform	Multiple platform	Multiple platform	Apple devices	NVIDIA GPU
Characteristic	Latency	Lightweight, modular, and scalable	Research	Large-scale deep neural networks	Memory footprint and power consumption.	Latency and throughput

for the cloud are not suitable for edge devices. Thus, to support data processing with deep learning models at the edge, several edge-based deep learning frameworks and tools have been released. In this section, we introduce TensorFlow Lite, Caffe2, PyTorch, MXNet, CoreML [70], and TensorRT [71], whose features are summarized in Table 4.

TensorFlow Lite [72] is TensorFlow's lightweight solution that is designed for mobile and edge devices. TensorFlow is developed by Google in 2016 and becomes one of the most widely used deep learning frameworks in cloud DCs. To enable low-latency inference of on-device deep learning models, TensorFlow Lite leverages many optimization techniques, including optimizing the kernels for mobile apps, prefused activations, and quantized kernels that allow smaller and faster (fixed-point math) models.

Facebook published Caffe2 [73] as a lightweight, modular, and scalable framework for deep learning in 2017. Caffe2 is a new version of Caffe, which is first developed by UC Berkeley AI Research (BAIR) and community contributors. Caffe2 provides an easy and straightforward way to play with the deep learning and leverage community contributions of new models and algorithms. Comparing with the original Caffe framework, Caffe2 merges many new computation patterns, including distributed computation, mobile, reduced precision computation, and more nonvision use cases. Caffe2 supports multiple platforms that enable developers to use the power of GPUs in the cloud or at the edge with cross-platform libraries.

PyTorch [68] is published by Facebook. It is a python package that provides two high-level features: tensor computation with strong GPU acceleration and deep neural networks built on a tape-based autograd system. Maintained by the same company (Facebook), PyTorch and Caffe2 have their own advantages. PyTorch is geared toward research, experimentation, and trying out exotic neural networks, while caffe2 supports more industrial-strength applications with a heavy focus on the mobile. In 2018, Caffe2 and PyTorch projects merged into a new one, named PyTorch 1.0, would combine the user experience of the PyTorch frontend with scaling, deployment, and embedding capabilities of the Caffe2 backend.

MXNet [69] is a flexible and efficient library for deep learning. It was initially developed by the University of Washington and CMU, to support convolutional neural network (CNN) and long short-term memory (LSTM) networks. In 2017, Amazon announced MXNet as its choice of the deep learning framework. MXNet places a special emphasis on speeding up the development and deployment of large-scale deep neural networks. It is designed to support multiple different platforms (either cloud platforms or the edge ones) and can execute training and inference tasks. Furthermore, other than the Windows, Linux, and OSX operating systems based devices, it also supports the Ubuntu Arch64 and Raspbian ARM-based operating systems.

CoreML [70] is a deep learning framework optimized for on-device performance at memory footprint and power consumption. Published by Apple, users can integrate the trained ML model into Apple products, such as Siri, Camera, and QuickType. CoreML supports not only deep learning models but also some standard models such as tree ensembles, SVMs, and generalized linear models. Built on the top of low-level technologies, CoreML aims to make full use of the CPU and GPU capability and ensure the performance and efficiency of data processing.

The platform of TensorRT [71] acts as a deep learning inference to run the models trained by TensorFlow, Caffe, and other frameworks. Developed by NVIDIA company, it is designed to reduce the latency and increase the throughput when executing the inference task on NVIDIA GPU. To achieve computing acceleration, TensorRT leverages several techniques, including weight and activation precision calibration, layer and tensor fusion, kernel autotuning, dynamic tensor memory, and multistream execution.

Considering different performances of the packages and the diversity of the edge hardware, it is challenging to choose a suitable package to build edge computing systems. To evaluate the deep learning frameworks at the edge and provide a reference to select appropriate combinations of package and edge hardware, pCAMP [74] is proposed. It compares the packages' performances (with respect to the latency, memory footprint, and energy) resulting from five edge devices and observes that no framework could win over all the others at all aspects. It indicates that there is much room to improve the frameworks at the edge. Currently, developing a lightweight, efficient, and high-scalability framework to support diverse deep learning modes at the edge cannot be more important and urgent.

In addition to these single-device-based frameworks, more researchers focus on distributed deep learning models over the cloud and edge. DDNN [75] is a distributed deep neural network architecture across cloud, edge, and edge devices. DDNN maps the sections of a deep

neural network onto different computing devices, to minimize communication and resource usage for devices and maximize the usefulness of features extracted from the cloud.

Neurosurgeon [76] is a lightweight scheduler that can automatically partition DNN computation between mobile devices and datacenters at the granularity of neural network layers. By effectively leveraging the resources in the cloud and at the edge, neurosurgeon achieves low computing latency, low energy consumption, and high traffic throughput.

C. Hardware System

The hardware designed specifically for deep learning can strongly support edge computing. Thus, we further review relevant hardware systems and classify them into three categories: field-programmable gate array (FPGA)-based hardware, GPU-based hardware, and application-specific integrated circuit (ASIC).

1) FPGA-Based Hardware: An FPGA is an integrated circuit and can be configured by the customer or designer after manufacturing. FPGA-based accelerators can achieve high-performance computing with low energy, high parallelism, high flexibility, and high security [77].

Zhang *et al.* [78] implement a CNN accelerator on a VC707 FPGA board. The accelerator focuses on solving the problem that the computation throughput does not match the memory bandwidth well. By quantitatively analyzing the two factors using various optimization techniques, the authors provide a solution with better performance and lower FPGA resource requirement, and their solution achieves a peak performance of 61.62 giga operations per second (GOPS) under a 100-MHz working frequency.

Following the above work, Qiu *et al.* [79] propose a CNN accelerator designed upon the embedded FPGA, Xilinx Zynq ZC706, for large-scale image classification. It presents an in-depth analysis of state-of-the-art CNN models and shows that convolutional layers are computational-centric and fully-connected layers are memory-centric. The average performances of the CNN accelerator at convolutional layers and the full CNN are 187.8 and 137.0 GOPS under a 150-MHz working frequency, respectively, which outperform previous approaches significantly.

An efficient speech recognition engine (ESE) is designed to speed up the predictions and save energy when applying the deep learning model of LSTM. ESE is implemented in a Xilinx XCKU060 FPGA operating at 200 MHz. For the sparse LSTM network, it can achieve 282 GOPS, corresponding to 2.52 tera operations per second (TOPS) on the dense LSTM network. In addition, energy efficiency improvements of $40\times$ and $11.5\times$ are achieved, respectively, compared with the CPU- and GPU-based solution.

2) GPU-Based Hardware: GPU can execute parallel programs at a much higher speed than CPU, which makes it fit for the computational paradigm of deep learning algorithms. Thus, to run deep learning models at the edge, building the hardware platform with GPU is a must choice. Specifically, NVIDIA Jetson TX2 and DRIVE PX2 are two representative GPU-based hardware platforms for deep learning.

NVIDIA Jetson TX2 [80] is an embedded AI computing device, which is designed to achieve low latency and high power efficiency. It is built upon an NVIDIA Pascal GPU with 256 CUDA cores, an HMP Dual Denver CPU, and a Qualcomm ARM CPU. It is loaded with 8 GB of memory and 59.7 GB/s of memory bandwidth and the power is about 7.5 W. The GPU is used to execute the deep learning task, and CPUs are used to maintain general tasks. It also supports the NVIDIA Jetpack SDK that includes libraries for deep learning, computer vision, GPU computing, and multimedia processing.

NVIDIA DRIVE PX [81] is designed as the AI supercomputer for autonomous driving. The architecture is available in a variety of configurations, from the mobile processor operating at 10 W to a multichip AI processors delivering 320 TOPS. It can fuse data from multiple cameras, as well as lidar, radar, and ultrasonic sensors.

3) ASIC: ASIC is the integrated circuit that supports customized design for a particular application rather than the general-purpose use. ASIC is suitable for the edge scenario as it usually has a smaller size, lower power consumption, higher performance, and higher security than many other circuits. Researchers and developers design ASIC to meet the computing pattern of deep learning.

DianNao family [82] is a series of hardware accelerators designed for deep learning models, including DianNao, DaDianNao, ShiDianNao, and PuDianNao. They investigate the accelerator architecture to minimize memory transfers as efficiently as possible. Different from other accelerators in DianNao family, ShiDianNao [83] focuses on image applications in embedded systems, which are widely used in edge computing scenarios. For the CNN-based deep learning models, it provides a computing speed $30\times$ faster than that of NVIDIA K20M GPU, with a body area of 4.86 mm^2 and a power of 320 mW.

Edge TPU [84] is Google's purpose-built ASIC for edge computing. It augments Google's Cloud TPU and Cloud IoT to provide an end-to-end infrastructure and facilitates the deployment of customers' AI-based solutions. In addition, Edge TPU can combine the custom hardware, open software, and state-of-the-art AI algorithms to achieve high performance with a small physical area and low power consumption.

VI. KEY DESIGN ISSUES

The edge computing system manages various resources along the path from the cloud center to end devices, shielding the complexity and diversity of hardware and helping developers quickly design and deploy novel applications. To fully leverage the advantages, we discuss the following

key issues that need to be paid attention when analyzing and designing a new edge computing system.

1) Mobility Support: Mobility support has two aspects: user mobility and resource mobility. User mobility refers to how to automatically migrate the current program state and necessary data when the user moves from one edge node coverage domain to another so that the service handover is seamless. Currently, Cloudlet and CloudPath provide service migration through terminating/finishing the existing task and starting a new VM/instance in the target edge node. Fine-grain migration and target edge node prediction are not supported. Resource mobility refers to: 1) how to dynamically discover and manage available resources, including both the long-term resources and short-term ones and 2) when some edge devices are damaged how the system can be resumed as soon as possible with replacements. For example, PCloud and FocusStack support edge devices dynamic join and leave with no task running. Intelligent dynamic resource management is still required for edge computing systems.

2) Multiuser Fairness: For edge devices with limited resources, how to ensure the fairness of multiuser usage, especially for the shared resources and rare resources, is important. For example, a smartphone made up of various sensors and computing resources can act as an edge node to serve multiple users. However, as the smartphone has limited battery life, it is a problem to fairly allocate resources when receiving many requests. The resource competition is more intense, the requests are coming from different irrelevant tasks, and it is hard to decide who should use the resource with only local information. In addition, the unfairness can be used as an attack strategy to make the critical task resource hungry, which may lead to the main task failure. The existing edge computing systems do not pay much attention to the multiuser fairness, the basic solution (bottom-line) is to provide resource isolation, and the users only get what the edge node promised when it accepts the request. More fairness strategies require system support, like related task status updates, and so on.

3) Privacy Protection: Unlike cloud computing, edge devices can be privately owned, such as gateway devices for smart home systems. When other users use such edge devices, obtain their data, and even take control of them, how to ensure the owner's privacy and guest users' data privacy is important. AirBox is a lightweight flexible EF system, which leverages hardware security mechanism (e.g, Intel SGX) to enhance system security. Other systems have not paid much attention to privacy and security. Existing cloud security approaches can be applied with the consideration of the resource limitation. Enhancing resource isolation, setting up privilege management and access control policies can be potential directions to solve the privacy problem.

4) Developer Friendliness: The system ultimately provides hardware interaction and basic services for upper level applications. How to design interactive APIs, program deployment module, resource application and revocation, and so on are the key factors for the system to be widely used. Therefore, to design an edge computing system, we should think from an application developer's perspective. Specifically, to provide effective development and deployment services is a good idea to help improve the ecosystem of the edge computing system. For instances, EdgeX Foundry and Apache Edgent provide APIs to manage devices and analyze data, respectively. ParaDrop supports monitoring devices and application status through developer APIs and provides a web UI for users to manage/configure gateway as well as deploy applications to devices.

5) Multidomain Management and Cooperation: Edge computing involves multiple types of resources, each of which may belong to a different owner, for example, the smart home gateways and sensors of the house owner, networks resources and base stations from the Internet service providers (ISP), and traffic cameras owned by the government. How to access these resources and organize them according to the requirements of application and services, especially in emergency situations, is a problem that the edge computing system needs to consider. Existing research studies assume we have permission to use various devices belongs to different owners. In the initial stage, systems focus on the functionality perspective rather than implementation issues like price model. With the developing of edge computing, the real deployment issues should be solved before we enjoy all the benefits.

6) Cost Model: In cloud computing, the corresponding VM can be allocated based on the resource requested by the user, and the cost model can be given according to the resource usage. In edge computing, an application may use resources from different owners. Thus, how to measure resource usage, calculate overall overhead, and give an appropriate pricing model are crucial problems when deploying an edge computing system. Generally, edge computing systems focus on how to satisfy the resource requirements of various services and applications, some of them pay more attention to the energy consumption due to the nature of mobile nodes, and more complex overhead are not considered.

7) Compatibility: Currently, specialized edge computing applications are still quite a few. For example, ParaDrop applications require an additional XML configuration file to specify the resource usage requirements and SpanEdge needs developers to divide/configure tasks into local tasks and global tasks. Common applications are not directly supported to run in edge systems. How to automatically and transparently convert existing programs to the edge version and conveniently leverages the advantages of edge computing are still open problems. The compatibility

should be considered in the edge system design, specifically how to adapt traditional applications to the new architecture and realize the basic functions so that they can run successfully, deserving further exploration and development.

VII. CONCLUSION

Edge computing is a new paradigm that jointly migrates the capability of networking, computation, and storage from the remote cloud to the user sides. Under the context of IoT and 5G, the vision of edge computing is promising in providing smarter services and applications with better user experiences. The recently proposed systems and tools on edge computing generally reduce the data processing and transmission overhead and improve the efficiency and efficacy of mobile data analytics. In addition, the integration of edge computing and deep learning techniques further fosters the research on edge-based intelligence services. This paper introduced the representative systems and open-source projects for edge computing, presented several energy-efficiency-enhancing strategies for performance consideration and technologies for deploying deep learning models at the edge, and suggested a few research directions during the system design and analysis. ■

REFERENCES

[1] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct./Dec. 2009.

[2] (2018). *Open Edge Computing*. [Online]. Available: http://openedgecomputing.org/

[3] Z. Li, C. Wang, and R. Xu, "Computation offloading to save energy on handheld devices: A partition scheme," in *Proc. Int. Conf. Compil., Archit., Synthesis Embedded Syst.*, 2001, pp. 238–246.

[4] K. Ha and M. Satyanarayanan, "Openstack++ for cloudlet deployment," School Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep., 2015.

[5] K. Ha, P. Pillai, W. Richter, Y. Abe, and M. Satyanarayanan, "Just-in-time provisioning for cyber foraging," in *Proc. 11th Annu. Int. Conf. Mobile Syst., Appl., Services*, 2013, pp. 153–166.

[6] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, "Towards wearable cognitive assistance," in *Proc. 12th Annu. Int. Conf. Mobile Syst., Appl., Services*, 2014, pp. 68–81.

[7] M. Satyanarayanan *et al.*, "Edge analytics in the Internet of Things," *IEEE Pervasive Comput.*, vol. 14, no. 2, pp. 24–31, Apr./Jun. 2015.

[8] M. Satyanarayanan, G. Lewis, E. Morris, S. Simanta, J. Boleng, and K. Ha, "The role of cloudlets in hostile environments," *IEEE Pervasive Comput.*, vol. 12, no. 4, pp. 40–49, Oct. 2013.

[9] S. H. Mortazavi, M. Salehe, C. S. Gomes, C. Phillips, and E. de Lara, "Cloudpath: A multi-tier cloud computing framework," in *Proc. 2nd ACM/IEEE Symp. Edge Comput.*, 2017, p. 20.

[10] M. Jang, K. Schwan, K. Bhardwaj, A. Gavrilovska, and A. Avasthi, "Personal clouds: Sharing and integrating networked resources to enhance end user experiences," in *Proc. INFOCOM*, Apr./May 2014, pp. 2220–2228.

[11] M. Jang and K. Schwan, "STRATUS: Assembling virtual platforms from device clouds," in *Proc. IEEE Int. Conf. Cloud Comput. (CLOUD)*, Jul. 2011, pp. 476–483.

[12] P. Liu, D. Willis, and S. Banerjee, "ParaDrop: Enabling lightweight multi-tenancy at the network's extreme edge," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Oct. 2016, pp. 1–13.

[13] H. P. Sajjad, K. Danniswara, A. Al-Shishtawy, and V. Vlassov, "SpanEdge: Towards unifying stream processing over central and near-the-edge data centers," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Oct. 2016, pp. 168–178.

[14] Z.-W. Xu, "Cloud-sea computing systems: Towards thousand-fold improvement in performance per watt for the coming zettabyte era," *J. Comput. Sci. Technol.*, vol. 29, no. 2, pp. 177–181, Jan. 2014.

[15] R. T. Fielding and R. N. Taylor, *Architectural Styles and the Design of Network-based Software Architectures*, vol. 7. Irvine, CA, USA: Univ. California, 2000.

[16] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, "Cachier: Edge-caching for recognition applications," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2017, pp. 276–286.

[17] U. Drolia, K. Guo, and P. Narasimhan, "Precog: Prefetching for image recognition applications at the edge," in *Proc. 2nd ACM/IEEE Symp. Edge Comput.*, 2017, p. 17.

[18] B. Amento, B. Balasubramanian, R. J. Hall, K. Joshi, G. Jung, and K. H. Purdy, "FocusStack: Orchestrating edge clouds using location-based focus of attention," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Oct. 2016, pp. 179–191.

[19] K. Bhardwaj, M.-W. Shih, P. Agarwal, A. Gavrilovska, T. Kim, and K. Schwan, "Fast, scalable and secure onloading of edge functions using airbox," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Oct. 2016, pp. 14–27.

[20] Q. Zhang, X. Zhang, Q. Zhang, W. Shi, and H. Zhong, "Firework: Big data sharing and processing in collaborative edge environment," in *Proc. 4th IEEE Workshop Hot Topics Web Syst. Technol. (HotWeb)*, Oct. 2016, pp. 20–25.

[21] Q. Zhang *et al.*, "OpenVDAP: An open vehicular data analytics platform for CAVs," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2018, pp. 1310–1320.

[22] L. Liu, X. Zhang, M. Qiao, and W. Shi, "SafeShareRide: Edge-based attack detection in ridesharing services," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Oct. 2018, pp. 17–29.

[23] R. Trimananda, A. Younis, B. Wang, B. Xu, B. Demsky, and G. Xu, "Vigilia: Securing smart home edge computing," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Oct. 2018, pp. 74–89.

[24] I. Zavalyshyn, N. O. Duarte, and N. Santos, "HomePad: A privacy-aware smart hub for home environments," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Oct. 2018, pp. 58–73.

[25] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "LAVEA: Latency-aware video analytics on edge computing platform," in *Proc. 2nd ACM/IEEE Symp. Edge Comput. (SEC)*, Jun. 2017, p. 15.

[26] C.-C. Hung *et al.*, "Videoedge: Processing camera streams using hierarchical clusters," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Oct. 2018, pp. 115–131.

[27] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2016, pp. 1–9.

[28] J. Wang *et al.*, "Bandwidth-efficient live video analytics for drones via edge computing," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Oct. 2018, pp. 159–173.

[29] Y. Li and W. Gao, "MUVR: Supporting multi-user mobile virtual reality with resource constrained edge cloud," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Oct. 2018, pp. 1–16.

[30] (2018). *Akraino Edge Stack*. [Online]. Available: https://www.akraino.org

[31] (2018). *Cord*. [Online]. Available: https://www.opennetworking.org/cord

[32] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 3, pp. 1617–1634, 3rd Quart., 2014.

[33] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal, "NFV: State of the art, challenges, and implementation in next generation mobile networks (vEPC)," *IEEE Netw.*, vol. 28, no. 6, pp. 18–26, Nov./Dec. 2014.

[34] A. W. Manggala, Hendrawan, and A. Tanwidjaja, "Performance analysis of white box switch on software defined networking using open vswitch," in *Proc. Int. Conf. Wireless Telematics*, Nov. 2015, pp. 1–7.

[35] K. C. Okafor, I. E. Achumba, G. A. Chukwudebe, and G. C. Ononiwu, "Leveraging fog computing for scalable IoT datacenter using spine-leaf network topology," *J. Elect. Comput. Eng.*, vol. 2017, Art. no. 2363240, Apr. 2017.

[36] (2018). *EdgeX Foundry*. [Online]. Available: https://www.edgexfoundry.org

[37] (2018). *Apache Edgent*. [Online]. Available: http://edgent.apache.org

[38] (2018). *Azure IoT*. [Online]. Available: https://azure.microsoft.com/en-us/overview/iot/

[39] R. Morabito, "Virtualization on Internet of Things edge devices with container technologies: A performance evaluation," *IEEE Access*, vol. 5, pp. 8835–8850, 2017.

[40] T. Mastelic, A. Oleksiak, H. Claussen, I. Brandic, J.-M. Pierson, and A. V. Vasilakos, "Cloud computing: Survey on energy efficiency," *ACM Comput. Surv.*, vol. 47, no. 2, p. 33, 2015.

[41] V. Valancius, N. Laoutaris, L. Massoulié, C. Diot, and P. Rodriguez, "Greening the Internet with nano data centers," in *Proc. 5th Int. Conf. Emerg. Netw. Exp. Technol.*, 2009, pp. 37–48.

[42] S. Sarkar and S. Misra, "Theoretical modelling of fog computing: A green computing paradigm to support IoT applications," *IET Netw.*, vol. 5, no. 2, pp. 23–29, 2016.

[43] A. Feldmann, A. Gladisch, M. Kind, C. Lange, G. Smaragdakis, and F.-J. Westphal, "Energy trade-offs among content delivery architectures," in *Proc. 9th Conf. Telecommun. Internet Media Technol. Econ. (CTTE)*, Jun. 2010, pp. 1–6.

[44] F. Jalali, K. Hinton, R. Ayre, T. Alpcan, and R. S. Tucker, "Fog computing may help to save energy in cloud computing," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 5, pp. 1728–1739, May 2016.

[45] G. Lewis, S. Echeverría, S. Simanta, B. Bradshaw, and J. Root, "Tactical cloudlets: Moving cloud computing to the edge," in *Proc. Mil. Commun. Conf. (MILCOM)*, Oct. 2014, pp. 1440–1446.

[46] H. Dubey, J. Yang, N. Constant, A. M. Amiri, Q. Yang, and K. Makodiya, "Fog data: Enhancing telehealth big data through fog computing," in *Proc. ASE BigData SocialInformatics*, 2015, p. 14.

[47] A. Asnaghi, M. Ferroni, and M. D. Santambrogio, "DockerCap: A software-level power capping orchestrator for docker containers," in *Proc. IEEE Int. Conf. Comput. Sci. Eng. (CSE) IEEE Int. Conf. Embedded Ubiquitous Comput. (EUC) 15th Int. Symp. Distrib. Comput. Appl. Bus. Eng. (DCABES)*, Aug. 2016, pp. 90–97.

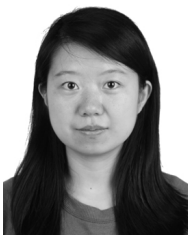
[48] T. Rausch, C. Avasalcai, and S. Dustdar, "Portable

energy-aware cluster-based edge computers," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Oct. 2018, pp. 260–272.
[49] Y. Nan *et al.*, "Adaptive energy-aware computation offloading for cloud of things systems," *IEEE Access*, vol. 5, pp. 23947–23957, 2017.
[50] C. Li *et al.*, "Towards sustainable in-situ server systems in the big data era," in *Proc. ACM SIGARCH Comput. Archit. News*, vol. 43, no. 3, pp. 14–26, 2015.
[51] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, 4th Quart., 2017.
[52] A. Rudenko, P. Reiher, G. J. Popek, and G. H. Kuenning, "Saving portable computer battery power through remote process execution," *ACM SIGMOBILE Mobile Comput. Commun. Rev.*, vol. 2, no. 1, pp. 19–26, 1998.
[53] R. Kemp *et al.*, "eyeDentify: Multimedia cyber foraging from a smartphone," in *Proc. 11th IEEE Int. Symp. Multimedia*, Dec. 2009, pp. 392–399.
[54] E. Cuervo *et al.*, "MAUI: Making smartphones last longer with code offload," in *Proc. 8th Int. Conf. Mobile Syst., Appl., Services*, 2010, pp. 49–62.
[55] K. Ha, "System infrastructure for mobile-cloud convergence," Ph.D. dissertation, Carnegie Mellon Univ., Pittsburgh, PA, USA, 2016.
[56] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proc. 6th Conf. Comput. Syst.*, 2011, pp. 301–314.
[57] M. V. Barbera, S. Kosta, A. Mei, and J. Stefa, "To offload or not to offload? The bandwidth and energy costs of mobile cloud computing," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 1285–1293.
[58] A. Fahim, A. Mtibaa, and K. A. Harras, "Making the case for computational offloading in mobile device clouds," in *Proc. 19th Annu. Int. Conf. Mobile Comput. Netw.*, 2013, pp. 203–205.
[59] W. Liu, T. Nishio, R. Shinkuma, and T. Takahashi, "Adaptive resource discovery in mobile cloud computing," *Comput. Commun.*, vol. 50, pp. 119–129, Sep. 2014.
[60] S. Abdelwahab, B. Hamdaoui, M. Guizani, and T. Znati, "Replisom: Disciplined tiny memory replication for massive IoT devices in LTE edge cloud," *IEEE Internet Things J.*, vol. 3, no. 3, pp. 327–338, Jun. 2016.
[61] S. Abdelwahab, B. Hamdaoui, M. Guizani, T. Znati, T. Leppnen, and J. Riekki, "Energy efficient opportunistic edge computing for the Internet of Things," *Web Intell. Agent Syst.*, to be published.
[62] P. P. Jayaraman, J. B. Gomes, H. L. Nguyen, Z. S. Abdallah, S. Krishnaswamy, and A. Zaslavsky, "CARDAP: A scalable energy-efficient context aware distributed mobile data analytics platform for the fog," in *Proc. East Eur. Conf. Adv. Databases Inf. Syst.* Cham, Switzerland: Springer, 2014, pp. 192–206.
[63] X. Zhang, Y. Wang, S. Lu, L. Liu, L. Xu, and W. Shi, "OpenEI: An open framework for edge intelligence," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2019.
[64] (2019). *AWS IoT Greengrass*. [Online]. Available: https://aws.amazon.com/greengrass/
[65] (2019). *Cloud IoT Edge: Deliver Google Ai Capabilities at the Edge*. [Online]. Available: https://cloud.google.com/iot-edge/
[66] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. OSDI*, vol. 16, 2016, pp. 265–283.
[67] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. 22nd ACM Int. Conf. Multimedia*, 2014, pp. 675–678.
[68] (2019). *From Research to Production*. [Online]. Available: https://pytorch.org
[69] T. Chen *et al.*, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," in *Proc. NIPS*, 2015.
[70] (2019). *Core ML: Integrate Machine Learning Models Into Your App*. [Online]. Available: https://developer.apple.com/documentation/coreml
[71] (2019). *Nvidia Tensorrt: Programmable Inference Accelerator*. [Online]. Available: https://developer.nvidia.com/tensorrt
[72] (2019). *Introduction to Tensorflow Lite*. [Online]. Available: https://www.tensorflow.org/mobile/tflite/
[73] (2019). *Caffe2: A New Lightweight, Modular, and Scalable Deep Learning Framework*. [Online]. Available: https://caffe2.ai/
[74] X. Zhang, Y. Wang, and W. Shi, "pCAMP: Performance comparison of machine learning packages on the edges," in *Proc. USENIX Workshop Hot Topics Edge Comput. (HotEdge)*, Boston, MA, USA, 2018. [Online]. Available: https://www.usenix.org/conference/hotedge18/presentation/zhang
[75] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2017, pp. 328–339.
[76] Y. Kang *et al.*, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGPLAN Notices*, vol. 52, no. 4, pp. 615–629, 2017.
[77] T. Wang, C. Wang, X. Zhou, and H. Chen, "A survey of FPGA based deep learning accelerators: Challenges and opportunities," 2018, *arXiv:1901.04988*. [Online]. Available: https://arxiv.org/abs/1901.04988
[78] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, 2015, pp. 161–170.
[79] J. Qiu *et al.*, "Going deeper with embedded FPGA platform for convolutional neural network," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, 2016, pp. 26–35.
[80] (2019). *NVIDIA Jetson TX2 Module*. [Online]. Available: https://developer.nvidia.com/embedded/buy/jetson-tx2
[81] (2019). *NVIDIA Drive PX: Scalable AI Supercomputer for Autonomous Driving*. [Online]. Available: https://www.nvidia.com/en-au/self-driving-cars/drive-px/
[82] Y. Chen, T. Chen, Z. Xu, N. Sun, and O. Temam, "Diannao family: Energy-efficient hardware accelerators for machine learning," *Commun. ACM*, vol. 59, no. 11, pp. 105–112, 2016.
[83] Z. Du *et al.*, "Shidiannao: Shifting vision processing closer to the sensor," *ACM Sigarch Comput. Archit. News*, vol. 43, no. 3, pp. 92–104, 2015.
[84] (2019). *Edge TPU: Google's Purpose-Built ASIC Designed to Run Inference at the Edge*. [Online]. Available: https://cloud.google.com/edge-tpu/

ABOUT THE AUTHORS

Fang Liu received the B.S. and Ph.D. degrees in computer science from the National University of Defense Technology, Changsha, China, in 1999 and 2005, respectively.

She is currently an Associate Professor with the School of Data and Computer Science, Sun Yat-sen University (SYSU), Guangzhou, China. Her current research interests include computer architecture, edge computing, and storage systems.



Youhuizi Li (Member, IEEE) received the B.E. degree in computer science from Xidian University, Xi'an, China, in 2010, and the Ph.D. degree in computer science from Wayne State University, Detroit, MI, USA, in 2016.

She is currently an Assistant Professor with the Key Laboratory of Complex Systems Modeling and Simulation, Ministry of Education, Hangzhou, China. She is also with the School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou. Her current research interests include energy efficiency, edge computing, and mobile system.

Dr. Li is a member of the China Computer Federation.

Guoming Tang received the bachelor's and master's degrees from National University of Defense Technology (NUDT), Guangzhou, China, in 2010 and 2012, respectively, and the Ph.D. degree in computer science from the University of Victoria, Victoria, BC, Canada, in 2017.

He is currently an Assistant Professor with the Key Laboratory of Science and Technology on Information System Engineering, NUDT. His current research interests include computational sustainability in edge/cloud computing systems aided by machine learning and optimization techniques.

Zhiping Cai (Member, IEEE) received the B.Eng., M.A.Sc., and Ph.D. degrees in computer science and technology from the National University of Defense Technology (NUDT), Guangzhou, China, in 1996, 2002, and 2005, respectively.

He is currently a Full Professor with the College of Computer, NUDT. His current research interests include network security and big data.

Dr. Cai is a Senior Member of the China Computer Federation. His Ph.D. dissertation received the Outstanding Dissertation Award of the Chinese PLA.

Xingzhou Zhang received the bachelor's degree from Shandong University, Jinan, China, in 2014. He is currently working toward the Ph.D. degree at the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China.

His current research interests include edge computing, machine learning, and computer systems.

Tongqing Zhou received the bachelor's and master's degrees in computer science and technology from the National University of Defense Technology (NUDT), Changsha, China, in 2012 and 2014, respectively, where he is currently working toward the Ph.D. degree at the College of Computer.

His current research interests include wireless networks, mobile sensing, and network measurements.