CampEdge: Distributed Computation Offloading Strategy Under Large-Scale AP-Based Edge Computing System for IoT Applications

Zhong Wang, *Graduate Student Member, IEEE*, Guangtao Xue, *Member, IEEE*, Shiyou Qian, *Member, IEEE*, and Minglu Li, *Senior Member, IEEE*

***Abstract*—With the development of multiaccess edge computing (MEC) technology at the network edge, efficient resources allocation and offloading between the resource-constrained edge clouds to maintain load balancing become a looming problem recently. However, most existing researches aimed at optimizing the allocation of computing resources are based on simulation and apply only for some typical Internet-of-Things (IoT) applications on mobile devices (i.e., they all lack of practicality and generality). In this article, we present a novel edge computing platform (CampEdge) with 36 edge nodes based on the wireless access point (AP) for adaptive resources allocation and computation offloading in a complicated dynamic campus environment. We first collected and sufficiently analyzed a large real-world WiFi data set, covering more than 8500 wireless APs and serves 44 000 active end users within an area of 3.1 km^2 over three months. A multiclass classification algorithm based on the random forest was then used with this data to accurately predict the state of resources usage at each edge node (i.e., in a busy state or normal state). To reasonably offload and transmit end-users' computational tasks to these edge nodes and optimize the total latency cost among the whole offloading process, we then illustrate a distributed computation offloading optimization strategy to formulate this complicated problem as a multiobjective latency optimization problem based on alternating direction method of multipliers (ADMM). Furthermore, we briefly discuss the convergence of our system. In experiments, CampEdge was shown to decrease user latency by up to 30%, compared to state-of-the-art methods. The proposed strategy was also shown to adaptively conform to a variety of compute-intensive and time-sensitive IoT applications for end users.**

***Index Terms*—Alternating direction method of multipliers (ADMM), computation offloading, Internet of Things, multiaccess edge computing (MEC), random forest (RF).**

Manuscript received April 13, 2020; revised July 13, 2020 and August 18, 2020; accepted September 20, 2020. Date of publication September 25, 2020; date of current version April 7, 2021. This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB2101102; in part by the Joint Key Project of the NSFC under Grant U1736207; in part by the Program of Shanghai Academic Research Leader under Grant 20XD1402100; and in part by the China Scholarship Council under Grant 201806230104. *(Corresponding author: Guangtao Xue.)*

Zhong Wang is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: zhongwang@sjtu.edu.cn).

Guangtao Xue and Shiyou Qian are with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: gt_xue@sjtu.edu.cn; qshiyou@sjtu.edu.cn).

Minglu Li is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China, and also with the College of Mathematics and Computer Science, Zhejiang Normal University, Jinhua 321004, China (e-mail: mlli@sjtu.edu.cn).

Digital Object Identifier 10.1109/JIOT.2020.3026862

I. Introduction

MULTIACCESS edge computing (MEC) [1], also referred to as fog computing [2] in some cases, is an emerging technology for 5G wireless communication systems and Internet-of-Things (IoT) technology [3] since it was standardized in 2015. It offers developers cloud-computing environment and capabilities at the edge of the network like mobile base station (BS) or wireless access point (AP) to meet currently high-speed mobile data transfer and low-latency customer service requests. Thus, end users can achieve real time and low-latency computation offloading among edge clouds (edge nodes). At present, MEC can be realized in a wide variety of IoT applications [4], [5], including video streaming and data analytics for unmanned aerial vehicles (UAVs), virtual reality (VR) and augmented reality (AR), location tracking services for autonomous vehicles (VANETs), face recognition for security payment, real-time monitoring for wearable devices, and so on. There are two main reasons for computation offloading for end users: 1) The computing capacity of mobile devices is insufficient to support some compute-intensive applications, such as high definition video analytics and 2) performing compute-intensive tasks locally (i.e., by energy-constrained mobile devices) leads to excessive energy consumption and performance degraded because the CPU frequency slows down to avoid overheating. Thus, it is essential to realize high-efficiency resources allocation and computation offloading for edge nodes.

A number of researches [6]–[14] focused on the optimization of edge computing resources in the past few years. However, previous attempts have run into two major limitations: 1) *lack of practicality*: most existing simulation-based platforms are inapplicable to most real-world scenarios and 2) *lack of generality*: most existing solutions are applicable to a limited number of IoT applications, i.e., they are not generalizable to compute intensive as well as time-sensitive applications. Furthermore, many offloading algorithms [11]–[16] involved the user in complex offloading processes and cost

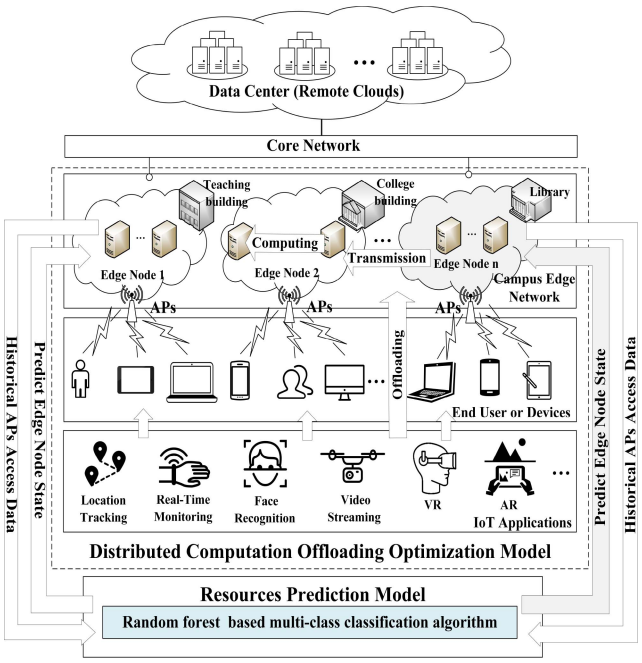


Fig. 1. Brief overview of CampEdge computing architecture. The resources prediction model predicts the edge node state (busy or normal) based on the historical APs access data. The distributed computation offloading optimization model contains three parts: 1) offloading; 2) transmission (if necessary); and 3) computing.

much user decision time, which tend to render the schemes less user friendly. For instance, the adaptive user-managed service placement (AUSP) mechanism in [11] focused on personalized service performance issues. It is only suitable for some compute-intensive IoT applications and cannot work well to reduce the computing and communication time. Meng *et al.* proposed an online dispatching and scheduling (Dedas) algorithm in [12] to schedule both networking bandwidth and computing resources. It is effective to decrease task completion time while can only be optimized for some time-sensitive applications. Ho and Jo [13] proposed an AP-based energy harvesting offloading model and a queueing model to maximize channel throughput using unlicensed bands, while it applies only to some band-limited applications. Given the above questions, an adaptive and effective edge computing system is extremely important for user experience in real life.

In this article, we monitored a GB-level real-world WiFi data set on a university campus over several months to identify the practical requirements for the allocation and offloading of edge cloud resources and facilitate the development of an effective edge computing system for IoT applications. Specifically, we consider a scenario in which a group of APs in certain places are aggregated as edge nodes to provide the computing capacity (AP loads) required for multiple resource-hungry end users and mobile devices. These edge nodes were deployed in some typical high-demand areas, such as the dining hall, the library, the teaching building, and the college building. Fig. 1 presents a brief overview of the proposed AP-based campus edge (CampEdge) computing architecture. This proposed platform includes two major models: 1) the multiclass classification computing resources prediction model and 2) the distributed computation offloading optimization model (we give the detailed illustration of these two models in Section III and Fig. 6). Our first objective was to obtain accurate predictions of the AP loads at each edge node. This was achieved using a multiclass classification algorithm based on the random forest (RF) in which edge nodes are divided into two classes: 1) busy state ($0 \leq$ average_AP_access_number ≤ 50) and 2) normal state (average_AP_access_number > 50). Statistical analysis of the WiFi data set revealed periodic variations in AP loads at these nodes, which could be predicted within certain periods like *mid-semester* and *summer vacation*. The second and most important objective was optimizing the utilization of computation resources among multiple edge nodes and offloads a variety of computational tasks to the most suitable node if necessary. We divide the offloading process into three parts (offloading, transmission, and computing) while taking into account three typical latency costs: 1) *offloading latency* (between end users and edge nodes); 2) *transmission latency* (between edge nodes if necessary); and 3) *computing latency* (within edge nodes). We then propose a distributed computation offloading optimization strategy to formulate the complicated offloading problem as a multiobjective optimization problem and use the optimized alternating direction method of multipliers (ADMM) algorithm to minimize the overall latency.

To the best of our knowledge, this is the first work to address edge cloud resources allocation and computation offloading problems based on the real-world large-scale wireless AP status data-trace in the public environment like the campus. The main achievements of this article are as follows.

1) We analyze a large-scale real-world WiFi data set containing the status of more than 8500 wireless APs and severs for 24,000 active end users in the campus covering an area of 3.1 km^2. More than 100 GB of data was collected over a period of three months from May 1, 2019 to July 28, 2019.
2) We present a novel adaptive edge computing platform (CampEdge) with 36 AP-based edge nodes for resources allocation and computation offloading based on the data trace. We analyze the AP load at each edge node at different times and then used an RF-based multiclass classification resources prediction model to forecast the usage of computing resources at each edge node precisely.
3) We formulate the offloading problem as a multiobjective optimization problem using a distributed optimization mechanism based on ADMM. The proposed CampEdge can serve both compute-intensive and time-sensitive IoT applications efficiently, while rendering all offload processes transparent to the end user.
4) We compare CampEdge with the state-of-the-art methods, it proved highly efficient at satisfying task requests for a range of applications at various time points, with up to a 30% decrease in user latency.

The remainder of this article is organized as follows. We extract and analyze the load features of the AP-based edge nodes in Section II. In Section III, we describe the system

TABLE I
NOTATION IN THIS ARTICLE

Notation	Definition of Notation
$\mathcal{M}$	the set of all end-users in campus
$\mathcal{N}$	the set of edge nodes in campus
K	the number of all APs in each edge node
$\mathcal{T}$	the time slot of a single day
$AP_{n,k}^{(t)}$	load of the k^{th} AP in node n at time slot t
$\overline{AP}_n$	the average AP loads of edge node n
L_{of}	offloading latency between end-user and edge node
L_{tr}	transmission latency for end-user's tasks
L_{cp}	computing latency of edge nodes for end-user's tasks
L_{all}	the overall latency for an offload process
λ	the AP loads state of edge node
ω	the weighting factor for each latency
x	the dual variable of λ and ω

model of the RF-based resources prediction algorithm and ADMM-based distributed computation offloading algorithm in detail, and we also discuss the convergence of the system. The performance of the proposed platform is then evaluated in Section IV. Section V shows the related works of MEC computation offloading in recent years. Conclusions and future works are described in Section VI.

II. System Description and Feature Analysis

A. System Overview

In order to understand realistic AP load features and design informed resources prediction and computation offloading strategy, it is essential to study empirical data in terms of distribution and variation in our large-scale WiFi system. In this section, we extract and analyze the load features of edge nodes in our CampEdge platform. We focused on this edge computing system in which the edge nodes are deployed based on thousands of wireless APs throughout the campus buildings. As shown in Fig. 1, there are $n \in \mathcal{N} = \{1, 2, \ldots, N\}$ edge nodes in well-frequented areas of the campus (e.g., dining hall, college building, library, and teaching building), where $m \in \mathcal{M} = \{1, 2, \ldots, M\}$ indicates the number of end users on campus (e.g., students, professors, and staff) who access the wireless APs in the above-mentioned locations with different types of IoT applications. Table I shows the notations we used in this article.

B. Data Collection and AP Description

We collect all end-users' association data and AP status data from our campus WiFi system. We use Python to implement the data collection program to call the data management APIs from the network management platform, which generates AP statistics in real time. To better understand this data set, we selected a typical AP from an edge node as an example. Table II lists the major AP and client parameters of a typical AP on campus. AP_detail includes, but not limited to, AP ID, group ID, AP name, SSID, device category, LAN IP/MAC, client count, and client list. AP ID refers to the number of the given AP and group ID refers to the number of the building deploying this AP. SSID refers to the service set identifier name assigned to a wireless network. Note that

Fig. 2. Scenario of several APs that deployed in our college building. It contains our laboratory, leisure room, lounge room, and multifunction room.

there are several types of SSID, such as CMCC, CMCC-EDU, SJTU, and SJTU-GUEST on our campus. AP name refers to named as the format of "MH-D4CY-2F-04," where "MH" is the abbreviation of our campus, "D4CY" is the abbreviation of the building (the fourth dining hall in this example), "2F" indicates the second floor of the building and "04" refers to the index tag. Client count indicates the user access number on one particular day and the client list is a historical list of these users with client_detail. Besides, we classified the APs into three types: 1) *thin AP* ($0 \leq$ client_number ≤ 10); 2) *medium AP* ($10 <$ client_number ≤ 50); and 3) the *busy AP* (client_number ≥ 50) based on different AP load on a particular day. This is the meaning of the device category. LAN IP and MAC, respectively, refer to the AP IP/MAC address, the details of which are not included for the sake of privacy. The client_detail of the AP contains client ID, user name, association status, connect and disconnect time, client MAC, IPv4 address, IPv6 address, authentication type, device type, and vendor. As shown in Table II, association status denotes the status property that whether the client is currently associated with the AP (True or False), connect and disconnect time are the timestamp records indicating the time of client log in and log out the AP (a 5-min interval in the example), authentication type refers to the WiFi access protocol to secure wireless networks (i.e., WPA, WPA-2, and WPA-3), and device type (e.g., Windows 10, iOS, Android, etc.) and vendor (e.g., Microsoft Corporation, Apple, Huawei, etc.) refer to the type of operating system and device manufacturers, respectively.

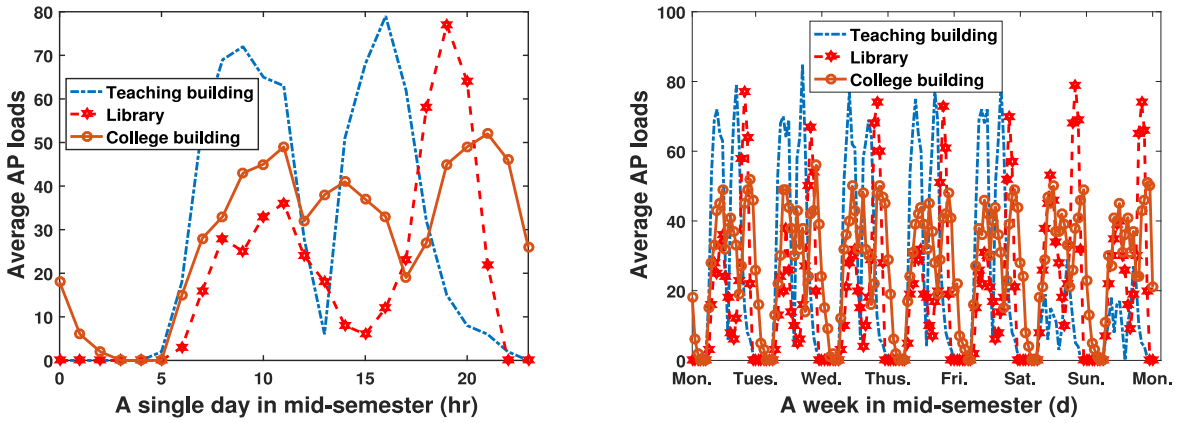
C. AP-Based Edge Nodes Feature Extraction and Analysis

There are more than 8500 wireless APs spread throughout the various campus buildings in the area of more than 3.1 km^2. Each building has a certain amount of APs. Fig. 2 is an example of several APs deployed in different places (e.g., our laboratory, leisure room, lounge room, and multifunction room) of our college building. The amount of APs in different building are different, thus the total APs loads in each building are also different. We cannot simply use these loads as the resources utilization of each building and compare them with each other. While we find that the average traffic consumption of a group of APs is more stable, so, in

TABLE II
MAJOR PARAMETERS OF THE AP ON A TYPICAL EDGE NODE IN CAMPUS

AP_Detail							
AP ID	***Group ID***	***AP Name***	***SSID***	***Device Category***	***LAN IP/MAC***	***Client Count***	***Client List***
317	004	MH-D4CY-2F-04	SJTU	Medium AP	NG[a]	35	NG[a]
Client_Detail							
Client ID	***User Name***	***Association Status***	***Connect Time***		***Disconnect Time***		***Client MAC***
206560669	Alice[a]	False	2019-05-18 T11:52:42 +08:00		2019-05-18 T11:57:47 +08:00		NG[a]
IPv4 Addr.	***IPv6 Addr.***	***Auth. Type***	***Device Type***		***Device Vendor***		
NG[a]	NG[a]	WPA-2	Windows 10		Microsoft Corporation		

[a]We hide the real information to ensure the personal privacy.



(a) (b)

Fig. 3. Average AP loads in three typical buildings during a (a) single day and (b) week in mid-semester.

this article, we classified the average AP load based on the number of AP accesses by end users or devices in a single day as follows: *light load* ($0 \leq$ client_number ≤ 10), *medium load* ($10 <$ client_number ≤ 50), and the *heavy load* (client_number > 50). Due to the large-scale data set of wireless APs status on campus, features were extracted during two periods: 1) *mid-semester* (two months from May 1, 2019 to June 30, 2019) and 2) *summer vacation* (one month from July 1, 2019 to July 28, 2019) for the model. In selecting edge nodes, we focused our analysis on buildings with distinctive AP load features. We sought to elucidate differences among edge nodes in a single working day and throughout the week based on feature sets from three typical edge nodes in three buildings: 1) *teaching building*; 2) *college building*; and 3) *library* as an example. Data sampling was conducted at a frequency of 1 h. So that we can finish the data collection of all APs after experimental testing. The time interval was denoted as $t \in \mathcal{T} = \{1, 2, \ldots, T\}$ in a single day. We assume that there are $k \in [1, K]$ APs in edge node n and that the load of each AP during time t was $\mathrm{AP}_{n,k}^{(t)}$. Thus, the AP loads AP_n of all APs at edge node n during a single day is calculated as follows:

$$\begin{aligned}\mathrm{AP}_n &= \mathrm{AP}_{n,1}^{(1)} + \mathrm{AP}_{n,2}^{(1)} + \cdots + \mathrm{AP}_{n,K}^{(1)} \\ &\quad + \cdots + \mathrm{AP}_{n,1}^{(T)} + \mathrm{AP}_{n,2}^{(T)} + \cdots + \mathrm{AP}_{n,K}^{(T)} \\ &= \sum_{k=1}^{K} \sum_{t=1}^{T} \mathrm{AP}_{n,k}^{(t)}. \end{aligned} \tag{1}$$

From this we estimate the average AP load at each edge node $\overline{\mathrm{AP}_n}$ in a single day as follows:

$$\overline{\mathrm{AP}_n} = \frac{\mathrm{AP}_n}{k} = \frac{\sum_{k=1}^{K} \sum_{t=1}^{T} \mathrm{AP}_{n,k}^{(t)}}{k}. \tag{2}$$

Intuitively, the average AP load at each edge node reflects the traffic consumption and resources utilization on campus during different time intervals throughout the day.

1) Mid-Semester Features: Fig. 3(a) illustrates the average AP loads in three buildings over a single day during mid-semester. The feature difference between them is obvious. For instance, the average AP loads were all in *heavy load* during 08:00–12:00 and 14:00–17:00 in the teaching building. Because it is during normal class time and most students are gathering in this place, thus have a higher probability to access the APs there. Meanwhile, the average AP loads in the library were in *light load* or *medium load* during the same time intervals and in *heavy load* during 18:00–22:00 cause most students are move to this place for self-study after class. The average AP loads in the colleges building were basically under a *medium load* throughout the day, since most of the users are postgraduates and researchers, they always spend much time in the laboratory. Fig. 3(b) presents the average AP loads in the same three buildings over one week in mid-semester. The average AP loads in teaching buildings dropped markedly over the weekend, whereas those in the library increased. This is because the courses always aggregated in working days and there have few classes in the teaching building at weekends. Thus, some users are move to the library for reading or studying during these time intervals. In addition, the average AP

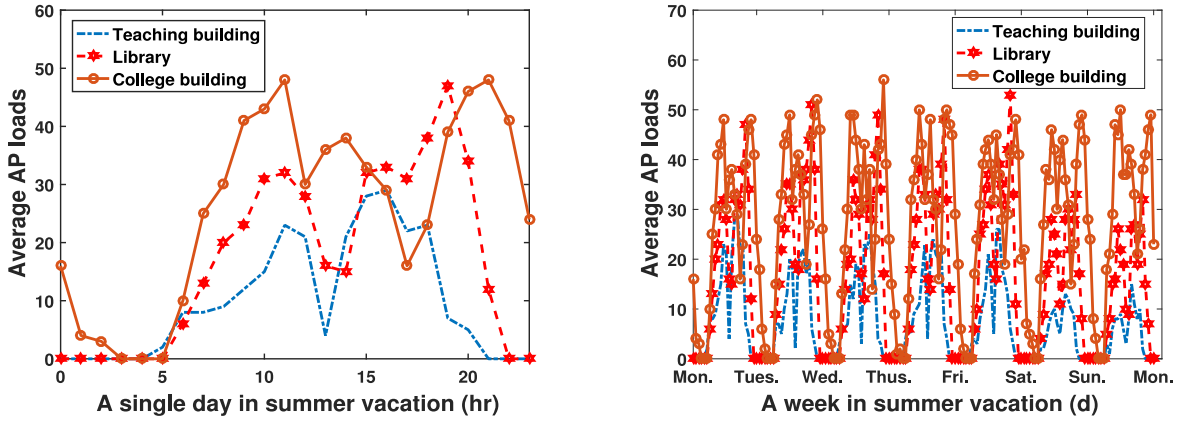


Fig. 4. Average AP loads in three typical buildings during a (a) single day and (b) week in summer vacation.

loads in college building basically remained at *medium load* throughout the entire week.

2) Summer Vacation Features: Edge node features during the summer vacation differed significantly from those obtained in *mid-semester*. As we can observe in Fig. 4, there is a visible drop in the average AP loads in teaching building and the library during the summer vacation. This can be attributed to the fact that many students return home over vacations if they have no courses in the summer semester (July 1–July 28). The average AP loads in the teaching building always were in *light load* throughout the week. While the average AP loads in the library are in *medium load* on weekdays and *light load* on weekends. Note that the AP loads in the college building during *mid-semester* differed little from those obtained during the *summer vacation*. This can be attributed to the fact that most postgraduates and researchers continue their research and experiment throughout the year (In other words, these people actually have no holidays in a whole year.)

It should also be noted that there was no campus WiFi coverage in students' dormitory of our university, thus we have no data set of these buildings. Nonetheless, it is reasonable that the average AP loads would drop dramatically during specific time intervals (e.g., 22:00–24:00 in a single day and 00:00–08:00 in the next day) during *mid-semester* and *summer vacation* because most students returned to dormitory from *teaching building* or *college building* during these periods.

III. Problem Formulation and System Model

This section presents a full description of the proposed CampEdge platform used to model the campus environment. We give a detailed elaboration of the resources prediction model and ADMM-based distributed computation offloading optimization model of CampEdge. Furthermore, we briefly explained the convergence of the system.

A. Resources Prediction and Allocation Model

Following the data analysis and feature extraction, a supervised learning algorithm in machine learning was used to predict the usage of computation resources at each edge node. In this article, the usage of computation resources actually is positively correlated to the AP loads at the AP-based edge node. Thus, a higher AP loads at one edge node is indicative of a larger number of end users or devices currently accessing this node and a corresponding increase in the need for computation resources in this node. So we developed a precise forecasting model to make a multiclass classification (i.e., predict the category of each edge node: *light load*, *medium load*, or *heavy load*) based on historical AP loads status. Edge nodes that are under high-load conditions present a major bottleneck in terms of resources allocation necessitating computation offloading. On the CampEdge platform, the edge nodes are then classified into two load states: 1) *heavy load* as the busy state and 2) the rest *light load* and *medium load* as the normal state. This renders the multiclass classification problem as a binary classification problem (one versus rest) to be addressed easily. For example, if we set a learner L^* as a training algorithm for binary classifiers and label the busy state sample as a positive sample ($y_1 = 1$) and the normal state as negative samples ($y_2 = 0, y_3 = 0$) from the AP loads samples set X, then we can obtain the following classifier model for the busy state samples:

$$\begin{cases} h_\theta^{(1)}(x) = P(y_1 = 1|x;\theta) \\ y_2 = 0, y_3 = 0. \end{cases} \tag{3}$$

Similarly, we can also set the normal state as positive samples ($y_2 = 2$ or $y_3 = 3$) and the busy state samples as negative samples ($y_1 = 0$) in order to obtain the classifier model for each sample, respectively

$$h_\theta^{(2)}(x) = P(y_2 = 2|x;\theta) \tag{4}$$

$$h_\theta^{(3)}(x) = P(y_2 = 3|x;\theta). \tag{5}$$

Thus, the following equation list all the classifier models for resources prediction:

$$h_\theta^{(j)}(x) = P(y_i = j|x;\theta) \tag{6}$$

where $i \in \{1, 2, 3\}, j \in \{1, 2, 3\}$. Algorithm 1 is a multiclass classification model for the prediction and allocation

Algorithm 1 Multiclass Classification Algorithm

Input:
The learner which is a training algorithm for binary classifiers, L^*;
The AP loads samples, X;
The labels y where $y_i \in \{1, \ldots, J\}$ is the label for the sample X_i, y.
Output:
A list of classifiers $h_\theta^{(j)}(x)$ for $j \in \{1, \ldots, J\}$, $h_\theta^{(j)}(x)$.

```
for each j ∈ {1, ..., J} do
    Construct a new label vector z;
    if z_i = j then
        z_i ← y_i;
    else
        z_i ← 0;
    end if
    // Apply L* to X, z to obtain h_θ^(j)(x).
end for
return h_θ^(j)(x)
```

of resources. It can be transformed into a binary classification problem to output several classifier models. Thus, when a new unseen sample x is input, the algorithm makes decisions by applying all of the above classifiers to identify the corresponding one with the highest confidence score, then predict the sample as the label j which this corresponding classifier has. The equation used for prediction is as follows:

$$\hat{y} = \underset{j \in \{1,\ldots,J\}}{\operatorname{argmax}} h_\theta^{(j)}(x). \tag{7}$$

We selected several optimized supervised learning algorithms as the classifier, including RF, naive bayes (NB), k-nearest neighbor (KNN), and decision tree (DT) for the prediction experiment using a small portion of typical APs from various time intervals in mid-semester and summer vacation. All of the algorithms have to be optimized and well suited for our platform. Then we use the precision–recall plot to measure the success of the prediction of each algorithm. We know that precision (P) is defined as the number of true positives (T_p) over the number of true positives plus the number of false positives (F_p). Recall (R) is defined as the number of true positives (T_p) over the number of true positives plus the number of false negatives (F_n). We have the functions as follows:

$$\begin{cases} P = \frac{T_p}{T_p + F_p} \\ R = \frac{T_p}{T_p + F_n}. \end{cases} \tag{8}$$

So, in our experiment, T_p refers to the number of correct classifications of the busy state nodes, F_p is the number of misclassified the normal state node as the busy state nodes, and F_n refers to the number of misclassified the busy state node as the normal state nodes (there also have a parameter of true negative T_n, which refers to the number of correctly classify of the normal state node). Then we achieve the values of precision and recall for each algorithm through the experiment and thus get the P–R plot. Fig. 5 presents the precision–recall plot obtained using these algorithms applied during *mid-semester* and *summer vacation*. We can find that the RF algorithm achieved the highest precision under a given recall level in both periods, with most of the edge node states

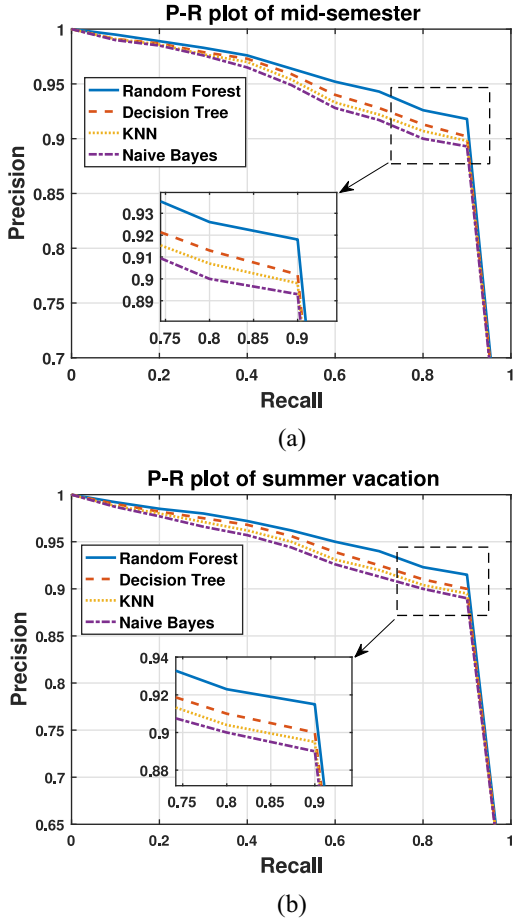


Fig. 5. Precision–recall plot of the aforementioned four types of supervised learning algorithms in mid-semester and summer vacation. (a) P–R plot of mid-semester. (b) P–R plot of summer vacation.

classified correctly in our platform. This means the RF algorithm is unexcelled in accuracy among current algorithms and the prediction model can achieve better performance based on this algorithm. This P–R plot method is enough to validate the precision of all the algorithms. Furthermore, the RF algorithm is also able to easily and efficiently handle a large WiFi data set compare to the other three algorithms, making it an ideal solution for resources prediction and allocation in our CampEdge platform.

B. Distributed Computation Offloading Model

The prediction model gave us an overall understanding of the resources usage expected at each edge node over the following few days. This provides the basis by which to optimize the utilization of resources between each edge node and transmit computational tasks from the busy state nodes to the normal state nodes (when possible). In this article, we assume that each end user or mobile device m can be served only by a single edge node n within a specified time interval t and all the computational tasks (no matter compute-intensive or time-sensitive applications) of end users are offloaded to the nearest edge node. Fig. 6 illustrates the resources prediction and computational tasks offloading process of the two major models in our CampEdge platform. As it is shown, the RF-based multiclass classification model

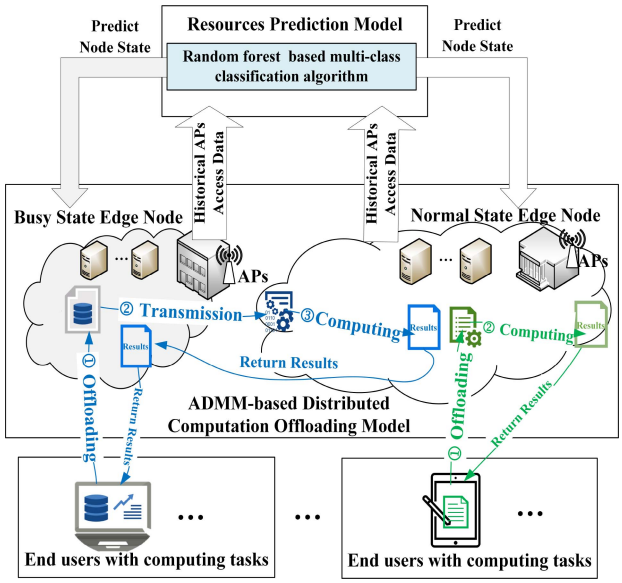


Fig. 6. Resources prediction and computational tasks offloading process of two major edge computing models.

can classify the edge node state as normal or busy based on the historical APs access data in the resources prediction process. In the ADMM-based distributed computation offloading model, we divided the computational tasks offloading process into three parts (offloading, transmission, and computing) while taking into account three typical costs in our platform: 1) *offloading latency* (between end users and edge nodes); 2) *transmission latency* (between edge nodes); and 3) *computing latency* (within edge nodes). For instance, if end-user's tasks are offloaded to the currently busy state edge node (which cause offloading latency ①), it should be transmitted by the optimization algorithm to one of the normal state edge nodes (thus lead to transmission latency ②), and compute on this node (which causes computing latency ③). On the contrary, if tasks are offloaded to the normal state edge node (which cause offloading latency ①), there have enough resources on this node to compute the tasks (which cause computing latency ②). Finally, the platform returns the results to the end user through the opposite path in both two situations. Our goal is to minimize these latencies as much as possible and use the distributed computation offloading model to address this latency optimization problem. Other costs like the results return latency and computation latency between APs within a single edge node was not taken into account because these two latencies can be negligible compare to the aforementioned three typical costs. We describe them in detail as follows.

1) Computing Latency: The term computing latency L_{cp} refers to the time cost associated with user's computational tasks at a given edge node. This latency pertains to two situations: *on-node* computing latency L_{on} and *out-node* computing latency L_{out}. On-node computing refers to a situation in which CampEdge platform selects the edge node that the user first accessed to perform computational tasks. There is a high probability that loads on this node is in the normal state. Out-node computing refers to a situation in which the CampEdge platform offloads and performs computational tasks to other suitable edge nodes when the node currently accessed is in the busy state. Both of these situations consume computation resources at related edge nodes and impose latency. This gives us the following equation:

$$\begin{cases} L_{cp} = \lambda_{n,t} L_{\text{on}} + \left(1 - \lambda_{n,t}\right) L_{\text{out}} \\ \lambda_{n,t} \in \{0, 1\} \quad \forall n, t \end{cases} \tag{9}$$

where $\lambda_{n,t}$ is the essential *state parameter* and denoting the computational tasks and AP load state of the edge node n during time interval t. Specifically, $\lambda_{n,t} = 1$ means that the node is in normal state and $\lambda_{n,t} = 0$ means it is in busy state.

2) Offloading Latency: The term offloading latency L_{of} indicates the time cost associated with offload user's computational tasks of different kind of IoT applications from the mobile devices of end user to the nearest edge node. This latency is closely associated with the current state of the edge node and the number of end users. Obviously, any decrease in computational load due to busy state edge node will significantly increase offloading latency between the end user and edge node. On the contrary, as long as the edge node is in normal state, then the offloading latency for the user will remain low. Likewise, more end user in edge node will lead to more offloading latency. From this, we can draw the following conclusion:

$$L_{of} \propto \frac{1}{\lambda_{n,t} m_{n,t}} \quad \forall n, t \tag{10}$$

where $m_{n,t}$ is the number of end users in edge node n within the time interval t. This means that there is a strong negative correlation between offloading latency and the node state parameter $\lambda_{n,t}$ and end-user number $m_{n,t}$. It actually fits well with the state of edge nodes.

3) Transmission Latency: As the name implies, transmission latency L_{tr} refers to the time cost associated with the transmission of computing tasks from a busy state edge node to a normal state edge node (as shown in Fig. 6). It is also closely associated with computing latency in the aforementioned two cases. On the one hand, if computational tasks are performed at the edge node itself, then there is no transmission latency, such that overall latency L_{all} includes only offloading latency and on-node computing latency L_{on}. On the other hand, if computational tasks must be transmitted, then the offload process will impose transmission latency and out-node computing latency L_{out}. Transmission latency can be denoted as follows:

$$\begin{cases} L_{tr} = 0, \left(\lambda_{n,t} = 1\right) \quad \forall n, t \\ L_{tr} \neq 0, \left(\lambda_{n,t} = 0\right) \quad \forall n, t. \end{cases} \tag{11}$$

This transmission latency is inevitable for busy state edge nodes, so our distributed offloading strategy needs to make a comprehensive consideration to find the perfect candidate node (with a few transmission latency and computing latency as possible) to transmit the computational tasks.

With the three costs mentioned above, we then tactfully transform the computation offloading problem into a distributed multiobjective latency optimization problem. That is

to say, we change a complicated unsolvable problem into an solvable practical problem. We use the weighted sum method to formulate the problem as follows:

$$\begin{aligned} \min \ & L_{\text{all}} = \sum_{n=1}^{N}\sum_{t=1}^{T} \omega_{n,t}^{(1)} L_{of} + \omega_{n,t}^{(2)} L_{cp} + \omega_{n,t}^{(3)} \big(1-\lambda_{n,t}\big) L_{tr} \\ \text{s.t.} \ & (2), (9)\text{–}(11) \end{aligned} \tag{12}$$

where $\omega_{n,t}^{(1)}$, $\omega_{n,t}^{(2)}$, and $\omega_{n,t}^{(3)}$ are the respective weighting factors for each cost, the value of which varies at different edge nodes n over different time intervals t; and $\delta_{n,t} = (1-\lambda_{n,t})$ is a parameter related to $\lambda_{n,t}$. Our objective is to optimize system performance by minimizing overall system latency L_{all} under the constraints of average AP loads of each edge node $\overline{\text{AP}_n}$ and the three typical latency costs.

ADMM is a widely used algorithm to deal with distributed convex optimization problems. It is well suited for our multiobjective optimization problem by breaking them into smaller pieces. So that each of them is then easier to manage. As mentioned previously, overall latency L_{all} is a function of the state parameter $\lambda \in \{\lambda_{n,t}\}$ and the weighting factor $\omega \in \{\omega_{n,t}^{(1)}, \omega_{n,t}^{(2)}, \omega_{n,t}^{(3)}\}$. Thus, the optimization problem can be transformed into the following equation:

$$\begin{aligned} \min_{\lambda,\omega} \ & f(\lambda) + g(\omega) \\ \text{s.t.} \ & A\lambda + B\omega = c. \end{aligned} \tag{13}$$

As this is a constrained minimization problem, where $f(\lambda)$ and $g(\omega)$ are functions with independent variables λ and ω, and A, B, and c are constants. We assume that f and g are convex. Thus, the optimal infimum value of this problem can be denoted by

$$p^{\star} = \inf\{f(\lambda) + g(\omega) | A\lambda + B\omega = c\}. \tag{14}$$

This the greatest lower bound of the model, and our objective is to approach this infimum value as much as possible. The method of multipliers is then used to transform (13) into the augmented Lagrangian function (using the scaled dual variable) as follows:

$$\begin{aligned} L_{\rho}(\lambda, \omega, x) = & f(\lambda) + g(\omega) + x^{T}(A\lambda + B\omega - c) \\ & + \frac{\rho}{2}\|A\lambda + B\omega - c\|_2^2 \end{aligned} \tag{15}$$

where x is the dual variable of λ and ω, ρ is the penalty factor (or dual update step size), and we have $\rho > 0$. This is an algorithm similar to dual ascent and method of multipliers. Thus, we obtain the iterations of each variable as follows:

$$\begin{cases} \lambda^{(j+1)} = \text{argmin}_{\lambda} L_{\rho}\big(\lambda, \omega^{(j)}, \quad x^{(j)}\big) \\ \omega^{(j+1)} = \text{argmin}_{\omega} L_{\rho}\big(\lambda^{(j+1)}, \quad \omega, x^{(j)}\big) \\ x^{(j+1)} = x^{(j)} + \rho\big(A\lambda^{(j+1)} + B\omega^{(j+1)} - c\big). \end{cases} \tag{16}$$

It contains a dual variable update step of x, a λ-minimization update step, and an ω-minimization update step in each iteration. In this article, we set a step size of the dual variable update of x which equals to the augmented Lagrangian parameter ρ in our ADMM model. So we will have the following simplified method of multipliers form for (13):

$$\begin{cases} \big(\lambda^{j+1}, \omega^{j+1}\big) = \text{argmin}_{\lambda,\omega} L_{\rho}\big(\lambda, \omega, x^{j}\big) \\ x^{(j+1)} = x^{(j)} + \rho\big(A\lambda^{(j+1)} + B\omega^{(j+1)} - c\big). \end{cases} \tag{17}$$

That is the augmented Lagrangian can be jointly minimized with the two primal variables λ and ω. We can minimize them sequentially through alternating minimization, wherein the ADMM-based computation offloading model is updated as follows:

$$\begin{cases} \lambda^{(j+1)} = \text{argmin}_{\lambda}\left(f(\lambda) + \frac{\rho}{2}\left\|A\lambda + B\omega^{(j)} - c + \frac{x^{(j)}}{\rho}\right\|_2^2\right) \\ \omega^{(j+1)} = \text{argmin}_{\omega}\left(g(\omega) + \frac{\rho}{2}\left\|A\lambda^{(j+1)} + B\omega - c + \frac{x^{(j)}}{\rho}\right\|_2^2\right) \\ x^{(j+1)} = x^{(j)} + \rho\big(A\lambda^{(j+1)} + B\omega^{(j+1)} - c\big). \end{cases} \tag{18}$$

The ADMM-based algorithm state consists of $\omega^{(j)}$ and $x^{(j)}$. It means that $(\omega^{(j+1)}, x^{(j+1)})$ is a function of $(\omega^{(j)}, x^{(j)})$, while $\lambda^{(j)}$ is not in the current state. It is achieved from the previous state of $(\omega^{(j-1)}, x^{(j-1)})$. Furthermore, if we define $\mu = (1/\rho)x$ as the scaled dual variable, we can also get the scaled form of the computation offloading model as follows:

$$\begin{cases} \lambda^{(j+1)} = \text{argmin}_{\lambda}\left(f(\lambda) + \frac{\rho}{2}\left\|A\lambda + B\omega^{(j)} - c + \mu^{(j)}\right\|_2^2\right) \\ \omega^{(j+1)} = \text{argmin}_{\omega}\left(g(\omega) + \frac{\rho}{2}\left\|A\lambda^{(j+1)} + B\omega - c + \mu^{(j)}\right\|_2^2\right) \\ \mu^{(j+1)} = \mu^{(j)} + A\lambda^{(j+1)} + B\omega^{(j+1)} - c. \end{cases} \tag{19}$$

Actually, this scaled form model is a scaled version of the dual variable x and it is shorter than the previous unscaled form model in (18). They are completely equivalent. So we typically use this scaled form in our proposed computation offloading model. We set $r^{(j)} = A\lambda^{(j)} + B\omega^{(j)} - c$ as the *primal residual* and $s^{j} = \rho A^{T} B(\omega^{j} - \omega^{j-1})$ as the *dual residual* at iteration j, then we can get the running sum of the residuals as

$$\mu^{(j)} = \mu^{(0)} + \sum_{k=1}^{j} r^{(k)}. \tag{20}$$

We can also have the following translation formula:

$$\begin{aligned} x^{T}r + (\rho/2)\|r\|_2^2 &= (\rho/2)\left\|r + (1/\rho)x\right\|_2^2 - (1/2\rho)\left\|x\right\|_2^2 \\ &= (\rho/2)\|r + \mu\|_2^2 - (\rho/2)\|\mu\|_2^2. \end{aligned} \tag{21}$$

So with an initial value of λ and ω and after j time iterations, we will get the most suitable iterative value and edge node to transmit and compute the computational tasks from the end user. Algorithm 2 is the pseudocode of the ADMM-based distributed offloading optimization model.

C. Convergence of the System

The convergence of ADMM-based system in each iteration of the algorithm can be validated [17] under the following two assumptions.

Assumption 1: The unaugmented Lagrangian L_0 has a saddle point.

Algorithm 2 ADMM-based Distributed Offloading Algorithm

Input:
The weighting factor $\omega \in \{\omega_{n,t}^{(1)}, \omega_{n,t}^{(2)}, \omega_{n,t}^{(3)}\}$, ω;
The state parameter $\lambda \in \{\lambda_{n,t}\}$, λ;
The constants A, B and c, and the penalty factor $\rho > 0$.
Output:
The iterative values $\lambda^{(j+1)}$, $\omega^{(j+1)}$ and $x^{(j+1)}$;
for each $j \in [1, J]$ **do**
  initialize a weighting factor ω, a state parameter λ and a dual variable x.
  $\lambda \leftarrow \lambda^{(0)}$;
  $\omega \leftarrow \omega^{(0)}$;
  $x \leftarrow x^{(0)}$.
  for $j = 1; j < J; j++$ **do**
    // Start the iterations;
    $\omega^{(j+1)} \leftarrow \arg\min_{\omega} L_{\rho}(\lambda^{(j+1)}, \omega, x^{(j)})$;
    $x^{(j+1)} \leftarrow x^{(j)} + \rho(A\lambda^{(j+1)} + B\omega^{(j+1)} - c)$;
    $\lambda^{(j+1)} \leftarrow \arg\min_{\lambda} L_{\rho}(\lambda, \omega^{(j)}, x^{(j)})$;
    Compute $\omega^{(j+1)}$;
    Update $x^{(j+1)}$;
    Update $\lambda^{(j+1)}$.
    // End the iterations.
  end for
  $\lambda \leftarrow \lambda^{(j+1)}$;
  $\omega \leftarrow \omega^{(j+1)}$;
  $x \leftarrow x^{(j+1)}$.
end for
return $\lambda^{(j+1)}$, $\omega^{(j+1)}$, $x^{(j+1)}$

Saddle point (or minimax point) is a point on the graph surface of a function where the slopes in orthogonal directions are all zero. For this assumption, we can express that there exists a saddle point $(\lambda^{\star}, \omega^{\star}, x^{\star})$, for which

$$L_0(\lambda^{\star}, \omega^{\star}, x) \leq L_0(\lambda^{\star}, \omega^{\star}, x^{\star}) \leq L_0(\lambda, \omega, x^{\star}) \tag{22}$$

holds for all λ, ω and x. This implies that $(\lambda^{\star}, \omega^{\star})$ is a solution to (13) and $x^{\star}$ is the dual optimal point, i.e., $A\lambda^{\star} + B\omega^{\star} = c$ and $f(\lambda^{\star}) < \infty, g(\omega^{\star}) < \infty$.

Assumption 2: Both functions of f and g are convex, proper, and closed.

We use the epigraphs of the functions to describe this assumption. The epigraph of a function is the set of points lying above or on its graph. A function is closed and convex if and only if its epigraph is a closed convex set. In this article, function f satisfies Assumption 2 if and only if its epigraph is a nonempty closed convex set

$$\text{epi } f = \{(\lambda, t) : \lambda \in \mathbf{R}^n, t \in \mathbf{R}, t \geq f(\lambda)\} \subseteq \mathbf{R}^{n+1} \tag{23}$$

and it is the same to the function g, which has a nonempty closed convex set of its epigraph

$$\text{epi } g = \{(\omega, t) : \omega \in \mathbf{R}^n, t \in \mathbf{R}, t \geq f(\omega)\} \subseteq \mathbf{R}^{n+1}. \tag{24}$$

Thus, our proposed ADMM-based distributed computation offloading model satisfies the following convergences under Assumptions 1 and 2.

1) Convergence of the Residual: The residual convergence means the final solution of the residual r is feasible and the *dual residual* s^j converges to zero after j time iterations, i.e.,

$$\begin{cases} r^j \to 0 \text{ as } j \to \infty \\ s^j = \rho A^T B(\omega^j - \omega^{j-1}) \to 0 \text{ as } j \to \infty. \end{cases} \tag{25}$$

2) Convergence of the Dual Variable: The final solution of dual variable will converge on a dual optimal point $x^{\star}$ after j time iterations, i.e.,

$$x^j \to x^{\star} \text{ as } j \to \infty. \tag{26}$$

3) Convergence of the Objective: This means the final objective function after j time iterations approaches optimal infimum value $p^{\star}$, i.e.,

$$f(\lambda^j) + g(\omega^j) \to p^{\star} \text{ as } j \to \infty. \tag{27}$$

All of these three system convergences can be proofed using the methods of Lyapunov function V^j through each iteration of the algorithm, which has

$$V^j = (1/\rho)\|x^j - x^{\star}\|_2^2 + \rho\|B(\omega^j - \omega^{\star})\|_2^2 \tag{28}$$

where V^j is a nonnegative quantity that depends on the values of $x^{\star}$ and $\omega^{\star}$ and decreases in each iteration. The detailed proof and derivation process of these convergences are too complicated, so we won't reiterate them here due to the space limitations of this article. In short, we can get the optimal adaptive computation offloading strategy with the minimum latency costs by using the ADMM-based distributed offloading model. Our experimental results demonstrate the validity of this algorithm.

IV. Evaluation of CampEdge Platform

In this section, we describe the numerical analysis used to evaluate the proposed CampEdge platform. All the evaluations were based on the real-world WiFi data set from our university campus. System performance was analyzed using several baseline metrics as well as typical time-sensitive and compute-intensive applications.

A. Experimental Setup

The prototype CampEdge was formulated using a real-world WiFi data set on campus constantly gathered over three months, with a focus on two time periods: 1) *mid-semester* (two months from May 1, 2019 to June 30, 2019) and 2) *summer vacation* (one month from July 1, 2019 to July 28, 2019) for the model. The status of all APs was established using a sampling frequency of 1 h, which resulted in 21 MB of status data in each round of data collection for all the APs. Note that over the duration of the study, this added up to more than 100 GB of data, indicating the status of 8500 APs serving 44 000 active end users within an area covering 3.1 km^2. Careful analysis of the data led us to establish 36 AP-based edge nodes, each of which was an aggregation of several APs. Most of the edge nodes are located in areas typical of a campus environment. In deploying the CampEdge platform, each edge node was assigned a fixed number of CPUs to provide computation and disk space for storage capacity for end users and mobile devices. The backbone network on the campus is a high-speed 10-GBE network and the standard building Ethernet connections across campus is 1 Gb/s, which was sufficient to ensure the required network bandwidth among the edge nodes. Fig. 7 presents the specific locations of the AP-based edge nodes of the CampEdge platform on our campus.

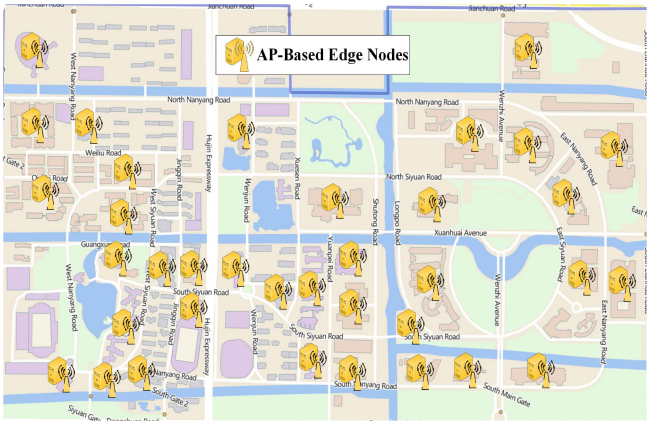


Fig. 7. Location of edge nodes in different buildings in our campus. Each edge node is aggregated by a number of APs.

B. Performance Baselines

The performance of the CampEdge platform was evaluated quantitatively using two typical resources scheduling and offloading strategies and a stable on-node strategy to provide as performance baselines. We then compared those values with the system performance obtained during *mid-semester* and *summer vacation*. The strategies assessed in this article were as follows.

1) AUSP Algorithm: This AUSP algorithm for MEC was proposed in [11]. This algorithm transforms the online service placement problem into a contextual multiarmed bandit (MAB) problem by treating the computation nodes as arms.

2) Dedas Algorithm: The other typical online deadline-aware task Dedas algorithm was proposed in [12]. This algorithm tries to dispatch the newly arrived task within a scheduled server based on the greedy scheduling algorithm. It also can choose to replace an existing task to meet the new deadline.

3) Remain on One Edge Node: The computing tasks remain on the edge node first accessed by the end user. Overall, this provides a stable on-node strategy suitable as baseline.

C. Evaluation of Resources Prediction Model

We sought to improve the accuracy of the resources prediction model by dividing the data sets in each time interval into the training set and testing set, respectively. The *mid-semester* data set was divided into a six-week training set (May 6–June 16) and a two-week testing set (June 17–June 30). The *summer vacation* data set was divided into a three-week training set (July 1–July 21) and a one-week testing set (July 22–July 28). Moreover, we use another three weeks' data set (from August 26–September 15, which contains a continuous period of time from summer vacation to the new semester) as testing set to further validate the prediction accuracy in a random manner (i.e., *random time periods*). After labeling all of the training data, we trained the RF-based multiclass classification prediction model for each edge node during the three time periods. All the training was based on the historical association status of the AP during the first six weeks of *mid-semester* and the first three weeks of *summer vacation*.

Fig. 8 illustrates the performance of our multiclass classification resources prediction model. We can see that the prediction accuracy always maintains at a high level in each time interval with an average accuracy of approximately 90.5%. Especially in *mid-semester* [Fig. 8(a)], the prediction accuracy over the time in two weeks (June 17–June 30) is superior to that for the one week (July 22–July 28) in *summer vacation* [Fig. 8(b)]. This is mainly due to the more end users and larger data set of average AP loads in each edge node in *mid-semester*. Our multiclass classification algorithm performs even better with a larger data set, giving a clear indication of the WiFi data set features of particular importance during classification. From Fig. 8(c), we can see that the model also indicates great prediction accuracy (over 90% on average) in the *random time periods*. It means that the prediction model can consistently keep at a high prediction accuracy over the long term, which guarantees the validity of the CampEdge platform. In addition, we can see that the model usually achieves a higher prediction accuracy on weekends than weekdays in both three time periods. That's presumably due to the limited users' mobility throughout the weekends, our model can be easier to predict AP loads in these relatively stable situations.

D. Evaluation of ADMM-Based Offloading Model

We selected three typical IoT applications to validate the performance of our proposed distributed computation offloading strategy.

1) Compute-Intensive Application: We choose the face recognition based on *OpenCV* [18] as a compute-intensive IoT application. We realize it by using Python to call the *OpenCV* APIs and run the recognition program on edge nodes.

2) Time-Sensitive Application: A time-sensitive IoT application of location tracking is used in our experiment. We realize this real-time tracking by using the *Gaode Navigation* [19] to navigate on our campus.

3) Mixed Demand Application: Mixed demand means that the IoT applications are both compute intensive and time sensitive, such as real-time video data analytics. We use a Google AR APP [20] to achieve it in our system.

Note that all of the experiments were repeated 50 times in the same situation, respectively, using the data sets of *mid-semester*, *summer vacation*, and *random time periods*. We then estimated the average overall latency of each algorithm.

As shown in Fig. 9, CampEdge has the lowest average overall latency, regardless of the time periods or applications. It can efficiently decreases end-users' latency by up to 30% compared to the other two algorithms (which have similar performance for all cases) according to statistics. This is due to the superior accuracy of the resources prediction model in the CampEdge platform, which allows for rapid response to end-users' computational task requests and the rapid offloading of tasks to the suitable edge node using the ADMM-based distributed optimization mechanism. Note that all three of the algorithms performed better in *summer vacation* than in the *mid-semester*. This is mainly because of the smaller number of end users and mobile devices on campus

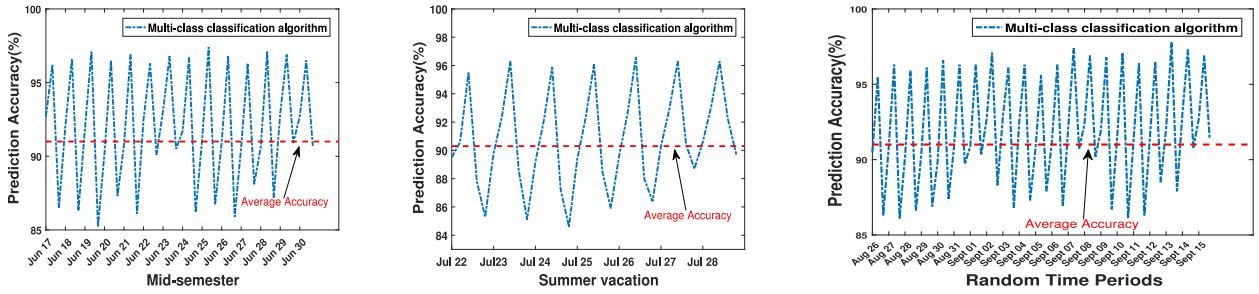


Fig. 8. Prediction accuracy of RF-based multiclass classification resources prediction model in different time periods and manner. (a) Prediction accuracy in mid-semester. (b) Prediction accuracy in summer vacation. (c) Prediction accuracy in random time periods.

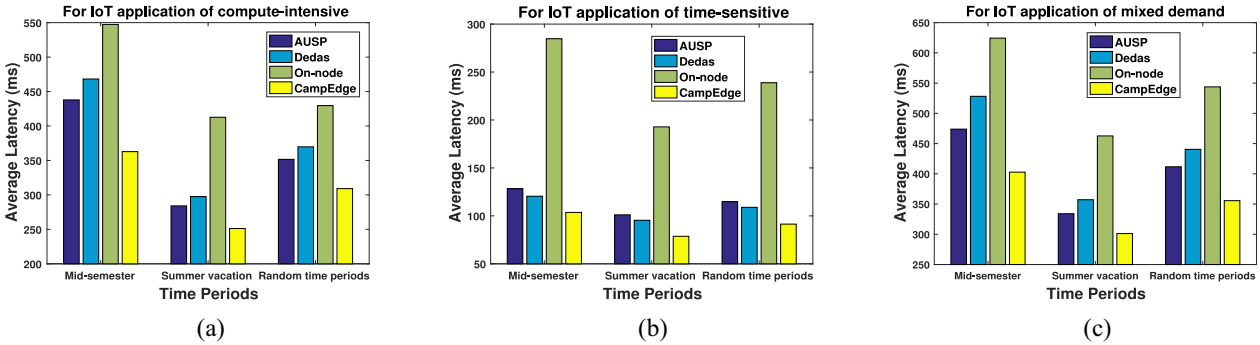


Fig. 9. Average overall latency for three typical applications in different time periods. (a)Average latency of compute-intensive application. (b) Average latency of time-sensitive application. (c) Average latency of mixed demand application.

during *summer vacation*. Edge clouds have more resources to be used to handle users' tasks. We also observe that the on-node method has the longest average overall latency, regardless of the three applications. This is an indication that users' mobility may have a profound influence on the user experience and system performance in MEC and it appears that computation offloading is necessary for this type of dynamic situation. Besides, we have verified the performance of the model with several other types of IoT applications (e.g., VR and HD video decoding). They also achieved great latency decreases compared to state-of-the-art methods. We will not go into the details due to the space limitations of this article.

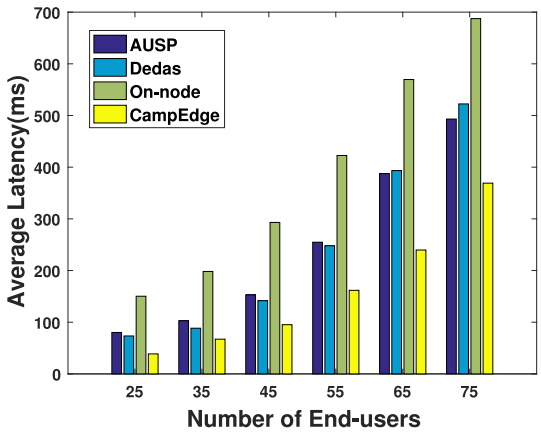


Fig. 10. Average overall latency for different number of end users.

E. Evaluation of System Convergence

We then, respectively, run experiments with the given number of end users or mobile devices $m \in \mathcal{M} = \{25, 35, 45, 55, 65, 75\}$ on a selected AP of an edge node to evaluate the average convergence time of the system and the average latency of the aforementioned algorithms. Note that these end users randomly use different kinds of IoT applications with the different number of computational tasks. This is a more realistic situation with different users' behaviors. The state of the AP on the edge node changes from normal state ($m = \{25, 35, 45\}$) to busy state ($m = \{55, 65, 75\}$) as the end-users' number m increases. We repeat all the experiments 50 times.

Fig. 10 demonstrates that the average latency of the aforementioned algorithms under the different number of end users. We can see that CampEdge achieves the lowest average latency in all of the algorithms and gets the system performance improvement by up to 35% on latency decreases for the end user. It proves the efficiency and applicability of our proposed ADMM-based distributed computation offloading optimization algorithm. Fig. 11 shows that the average convergence time of CampEdge system rises almost linearly as the end-users' number m increases. This indicates that our system can achieve rapid convergence with the ADMM-based distributed optimization mechanism in each iteration and have little fluctuation as the number of end users or computational tasks changes. It verifies the stability and reliability of the system under the complex dynamic environment on campus.

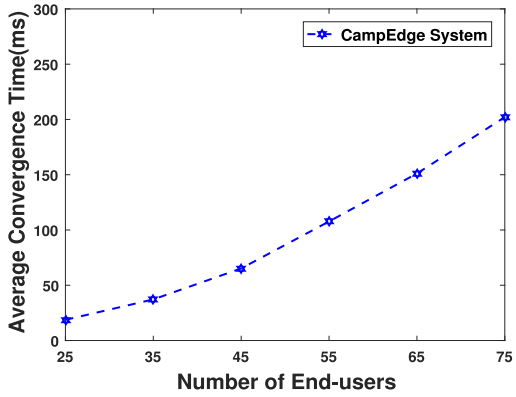


Fig. 11. Average convergence time for different number of end users.

V. Related Work

In 2015, ETSI [21] presented the first white paper of MEC with standardized definitions. Since that time, more and more research studies focus on this emerging technology in the past few years. One of the main topics has been the edge cloud resources allocation and the computation task offloading problems. For instance, Chen *et al.* [6] formulated the computation offloading decision-making problem as a multiuser computation offloading game and presented an efficient offloading algorithm to achieve the Nash Equilibrium. Xu *et al.* [7] proposed an efficient online algorithm to jointly optimize dynamic service caching and task offloading in the MEC system. Cicconetti *et al.* [8] proposed their low-latency distributed computation offloading strategy from the user terminal to edge devices for mobile pervasive environments.

There has been a significant amount of effort on the study of energy-efficient offloading mechanisms [9], [10], [22]–[26] for MEC. For example, You *et al.* [9] proposed an energy-efficient resources allocation system based on OFDMA and TDMA technology in wireless communication. Xiao and Krunz [10] sought a tradeoff between users' QoE and edge nodes' power efficiency and proposed a cooperation offload forwarding strategy to maximizes users' QoE under the given power efficiency. Li *et al.* [24] implemented the optimal task offloading strategy in a time-division manner for a predetermined execution order under the constraints of limited computation and communication resources. Note that most of these aforementioned studies (with their own relative advantages) were based on simulations, such that they may lack practical verification and largely inapplicable to a real-world scenario. Our edge nodes are based on the scenario of our campus WiFi data set and AP load. Those data traces were used in the development of our CampEdge system using resources allocation and computation offloading for MEC. Besides, many companies or research groups [27]–[31] have presented their CampEdges currently offered in the marketplace. For instance, *KubeEdge* [27] is an open-source system which built upon *Kubernetes* [32] and Eclipse *Mosquitto* [33] for extending native *Docker* [34] containerized applications orchestration capabilities to host at the edge. *Akama* [29] and other companies (HP, Amazon) proposed a platform for computing on the mobile edge in 2018. *Azion* [31] was established to enable customers to build and run the serverless applications right at the network edge, thus closer to end users and devices with low latency. However, these platforms still somewhat limited in terms of commercial applicability and are ill suited to the dynamic and complex environments, such as the campus addressed in this article.

Other studies [11]–[16] focused on optimizing performance for some specific applications. Ouyang *et al.* [11] focused on personalized service performance issues and propose an adaptive user-managed service optimization mechanism. This mechanism just suitable for some compute-intensive mobile applications and cannot work well to reduce the computing and communication time. Meng *et al.* [12] presented an online Dedas algorithm to schedule both networking bandwidth and computing resources. It is effective to decrease task completion time but they can only be optimized for the typical time-sensitive applications. Liu *et al.* [14] designed a system that enables high accuracy object detection and proposed several low-latency offloading techniques that reduce offloading latency and bandwidth consumption. While their offloading techniques are only suitable for AR and mixed reality (MR) systems. Yi *et al.* [15] described a computation offloading scheme, a pricing rule, and a transmission scheduling discipline-based MEC framework for multiuser. *Lavea* [16] is a platform that offloads computation between clients and nearby edge nodes to provide low-latency video analytics. These methods are only suitable for some delay-sensitive applications. Note that all of the above studies are user-managed. It may consume much user decision time on whether or not to offload its task and where to offload. A well-organized edge computing system should be simple, effective, and most of all user friendly. The proposed CampEdge system proved highly effective in all three of these areas and well suited for a variety of IoT applications.

VI. Conclusion

In this article, we present a novel adaptive resources allocation and distributed edge computing offloading platform based on a large volume of real-world wireless APs data set on campus. On the analysis of this practical data, we use a RF-based multiclass classification prediction algorithm to accurately monitor and predict the computing resource's usage of each edge node. Furthermore, we then use an ADMM-based distributed multiobjective latency optimization mechanism to address the computation offloading problem tactfully. Experimental results have shown that it can decrease user latency by up to 30%, compared to state-of-the-art methods. It also demonstrates the efficiency, generality, and applicability of our proposed CampEdge platform.

We realized that it is one of the particular scenarios for edge computing with distinctive AP load features of the campus and the end user are mainly students, teachers, and university staff. It has a good performance but may not universally applicable for all the other scenarios or conditions. For instance, the urban level CampEdge (i.e., serving an enormous volume of end users, vehicles, and mobile devices with different types) is a large-scale complex scene with real-time task requests. These dynamically changing task requests are flexible and diverse. They make it more difficult to realize an optimized

resources allocation algorithm and offloading mechanism. So a well-designed urban level CampEdge is important for the widespread use of 5G technology and could be a future research direction for us. Moreover, we may further capture and tracking a large body of data about end-user behavior in campus or city and make an optimized strategy to solve the global path planning problem.

REFERENCES

[1] ETSI. (2017). *Multi-Access Edge Computing (MEC)*. [Online]. Available: https://www.etsi.org/technologies/multi-access-edge-computing

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st ed. MCC Workshop Mobile Cloud Comput.*, 2012, pp. 13–16.

[3] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1657–1681, 3rd Quart., 2017.

[4] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[5] S. Chen, H. Xu, D. Liu, B. Hu, and H. Wang, "A vision of IoT: Applications, challenges, and opportunities with china perspective," *IEEE Internet Things J.*, vol. 1, no. 4, pp. 349–359, Aug. 2014.

[6] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.

[7] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2018, pp. 207–215.

[8] C. Cicconetti, M. Conti, and A. Passarella, "Low-latency distributed computation offloading for pervasive environments," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. (PerCom)*, 2019, pp. 1–10.

[9] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Trans. Wireless Commun.*, vol. 16, no. 3, pp. 1397–1411, Mar. 2017.

[10] Y. Xiao and M. Krunz, "QoE and power efficiency tradeoff for fog computing networks with fog node cooperation," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2017, pp. 1–9.

[11] T. Ouyang, R. Li, X. Chen, Z. Zhou, and X. Tang, "Adaptive user-managed service placement for mobile edge computing: An online learning approach," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2019, pp. 1468–1476.

[12] J. Meng, H. Tan, C. Xu, W. Cao, L. Liu, and B. Li, "Dedas: Online task dispatching and scheduling with bandwidth constraint in edge computing," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2019, pp. 2287–2295.

[13] J. Ho and M. Jo, "Offloading wireless energy harvesting for IoT devices on unlicensed bands," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 3663–3675, Apr. 2019.

[14] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *Proc. ACM Int. Conf. Mobile Comput. Netw. (MobiCom)*, 2019, pp. 1–16.

[15] C. Yi, J. Cai, and Z. Su, "A multi-user mobile computation offloading and transmission scheduling mechanism for delay-sensitive applications," *IEEE Trans. Mobile Comput.*, vol. 19, no. 1, pp. 29–43, Jan. 2020.

[16] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "Lavea: Latency-aware video analytics on edge computing platform," in *Proc. ACM/IEEE Symp. Edge Comput. (SEC)*, 2017, pp. 1–13.

[17] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, 2011.

[18] OpenCV. (2020). *OpenCV: Open Source Computer Vision Library*. [Online]. Available: https://opencv.org/

[19] (2020). *Autonavi*. [Online]. Available: https://www.autonavi.com/

[20] (2020). *GoogleAR*. [Online]. Available: https://arvr.google.com/ar/

[21] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing a key technology towards 5G," ETSI, Sophia Antipolis, France, White Paper, pp. 1–16, 2015.

[22] A. Al-Shuwaili and O. Simeone, "Energy-efficient resource allocation for mobile edge computing-based augmented reality applications," *IEEE Wireless Commun. Lett.*, vol. 6, no. 3, pp. 398–401, Jun. 2017.

[23] Z. Wang, D. Sun, G. Xue, S. Qian, G. Li, and M. Li, "Ada-things: An adaptive virtual machine monitoring and migration strategy for Internet of things applications," *J. Parallel Distrib. Comput.*, vol. 132, pp. 164–176, Oct. 2019.

[24] Y. Li, G. Xu, J. Ge, P. Liu, and X. Fu, "Energy-efficient resource allocation for application including dependent tasks in mobile edge computing," *KSII Trans. Internet Inf. Syst.*, vol. 14, no. 6, pp. 2422–2443, 2020.

[25] Y. Xiao and M. Krunz, "Distributed optimization for energy-efficient fog computing in the tactile Internet," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 11, pp. 2390–2400, Nov. 2018.

[26] Q. Zhang, Q. Zhang, W. Shi, and H. Zhong, "Firework: Data processing and sharing for hybrid cloud-edge analytics," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 9, pp. 2004–2017, Sep. 2018.

[27] KubeEdge. (2019). *Kubeedge, a Kubernetes Native Edge Computing Framework*. [Online]. Available: https://kubeedge.io/en/

[28] (2017). *EdgeMicro*. [Online]. Available: https://www.edgemicro.com/

[29] Akamai. (2019). *Akamai Is the Edge*. [Online]. Available: https://www.akamai.com/

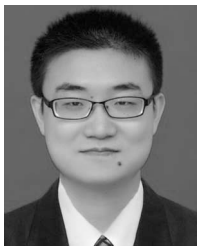
[30] Saguna. (2019). *Transforming Communication Networks Into Edge Cloud Computing Platforms*. [Online]. Available: https://www.saguna.net/

[31] Azion. (2018). *Build Scalable, Reliable and Secure Serverless Applications at the Edge*. [Online]. Available: https://www.azion.com/en/

[32] Kubernetes (K8s). (2020). *An Open-Source System for Automating Deployment, Scaling, and Management of Containerized Applications*. [Online]. Available: https://kubernetes.io/

[33] EclipseMosquitto. (2019). *An Open Source Message Broker That Implements the MQTT Protocol*. [Online]. Available: https://mosquitto.org/

[34] Docker. (2020). *Docker: The Modern Platform for High-Velocity Innovation*. [Online]. Available: https://www.docker.com/



Zhong Wang (Graduate Student Member, IEEE) received the master's degree from the College of Computer Science and Electronic Engineering, Hunan University, Changsha, China, in 2015. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China.

From September 2018 to September 2019, he was also a joint Ph.D. student under the guidance of Prof. Baochun Li, with the ECE, University of Toronto, Toronto, ON, Canada, which was funded by CSC. His research interests include cloud and edge computing and distributed machine learning.

Guangtao Xue (Member, IEEE) received the Ph.D. degree from Shanghai Jiao Tong University, Shanghai, China, in 2004.

He is currently a Professor with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, where he is also the Deputy Dean of the School of Electronic Information and Electrical Engineering. His research interests include wireless networks, big data, mobile computing, and distributed computing.

Prof. Xue is a member of IEEE Computer and the IEEE Communication Society.

Shiyou Qian (Member, IEEE) received the Ph.D. degree from Shanghai Jiao Tong University, Shanghai, China, in 2015.

He is currently an Associate Professor with the Department of Computer Science and Engineering, Shanghai Jiao Tong University. His research interests include event matching for content-based publish/subscribe systems, resource scheduling for hybrid cloud, and driving recommendation with vehicular networks.

Minglu Li (Senior Member, IEEE) received the Ph.D. degree from Shanghai Jiao Tong University, Shanghai, China, in 1996.

He is currently a Professor with the Department of Computer and Engineering, Shanghai Jiao Tong University. He is also a Professor with the College of Mathematics and Computer Science, Zhejiang Normal University, Jinhua, China. He is the Director of the IBM-SJTU Grid Research Center, Shanghai Jiao Tong University. His main research topics include big data, cloud and edge computing, and wireless sensor networks.