EBI-PAI: Toward an Efficient Edge-Based IoT Platform for Artificial Intelligence

Shu Yang, Kunkun Xu, Laizhong Cui, *Senior Member, IEEE*, Zhongxing Ming, Ziteng Chen, and Zhong Ming, *Member, IEEE*

***Abstract*—Edge computing, especially multiaccess edge computing, is seen as a promising technology to improve the Quality of user Experience (QoE) of many artificial intelligence (AI) applications in the evolution toward Internet-of-Things (IoT) infrastructure. However, the management and deployment of massive edge data centers bring new challenges for the current network. In this article, we propose a new edge-based IoT platform for AI (EBI-PAI), based on software-defined network (SDN) and serverless technology. EBI-PAI provides a unified service calling interface and schedules the resources automatically to satisfy the QoE requirements of users. To optimize performances during incremental deployment, we formulate the deployment problem, prove its complexity, and design heuristic algorithms to solve it. We implement EBI-PAI based on an opensource serverless project and deploy it in real networks. To evaluate EBI-PAI, we conduct comprehensive simulations based on the generated and real-world network topology, and real-world base station data set. The simulation results show that EBI-PAI can greatly improve QoE with the same budget and save the budget to achieve similar QoE. We finally carry out a case study with real user demands, and it further validates the simulation results.**

***Index Terms*—5G, incremental deployment, multiaccess edge computing (MEC).**

I. Introduction

RECENTLY, Quality of user Experience (QoE)-aware artificial intelligence (AI) applications have been growing actively with the Internet of Things (IoT) all around the world. For example, AI-based self-driving vehicles need to compute fast to guarantee safety [1]; AI-empowered cameras need high bandwidth to upload the captured videos to video processing servers; and AI-based remote control of equipment in the industrial IoT is delay sensitive, the communication delay is required to be less than 10 ms [2].

Manuscript received May 15, 2020; revised July 12, 2020; accepted August 9, 2020. Date of publication August 24, 2020; date of current version June 7, 2021. This work was supported in part by the National Key Research and Development Plan of China under Grant 2018YFB1800302 and Grant 2018YFB1800805; in part by the National Natural Science Foundation of China under Grant 61772345, Grant 61902258, Grant 61672358, and Grant 61836005; in part by the Major Fundamental Research Project in the Science and Technology Plan of Shenzhen under Grant JCYJ20190808142207420 and Grant GJHZ20190822095416463; and in part by the Pearl River Young Scholars Funding of Shenzhen University. *(Corresponding author: Laizhong Cui.)*

Shu Yang, Kunkun Xu, Zhongxing Ming, Ziteng Chen, and Zhong Ming are with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China.

Laizhong Cui is with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China, and also with Peng Cheng Laboratory, Shenzhen 518066, China (e-mail: cuilz@szu.edu.cn).

Digital Object Identifier 10.1109/JIOT.2020.3019008

Although computing power of end devices is growing, the gap between device power and user demands is not narrowing [3], so computation offloading is still required [4]. Within the current cloud-based network, it is difficult to meet the QoE requirements of these applications, because: 1) the cloud data centers are far away from users and 2) the Internet provides the best effort hop-by-hop services, and multihop communications between cloud and users bring additional latencies, especially during congestion conditions [5], [6].

In recent years, the IoT has become deeply ingrained in our society by transforming everyday objects, such as wearable devices, transportation, and augmented reality into communication devices, and has brought new challenges and opportunities [7]. Cisco data show that by 2020, more than 50 billion devices will be connected to the network, and it is clear that the current infrastructure will not be able to support all the generated data. It has the following drawbacks: 1) the delay and bandwidth are limited due to the distance between users and the cloud computing center and 2) it brings overheads to the infrastructure as the traffic travels through multiple links across the networks.

By placing micro data centers closer to users and reducing the round-trip delay between end-user devices and computing servers, edge computing is seen as a promising technology to solve the problems. It is also seen as the key to serving QoE-aware applications, such as achieving low-latency communications. Currently, edge computing is considered as an important direction for 5G, and many working groups are working on it to standardize 5G multiaccess edge computing (MEC) [8].

However, MEC brings more challenges for service providers: 1) operation and maintenance of a large number of micro and distributed data centers are more difficult than a few large and centralized data centers; 2) users can not manipulate the complexity of underlying infrastructures; and 3) the computing and bandwidth resources need to be carefully scheduled to satisfy the QoE requirements.

There are many IoT platforms [9] aiming at managing IoT networks, but it is difficult to find one that complies with the requirements of opensource and heterogeneity for the future IoT scenario. Matsuda *et al.* [10], Omnes *et al.* [11], and Bizanis and Kuipers [12] described the innovative technologies that will support the implementation of IoT architecture, such as the NFV, MEC, and management and orchestration (MANO), where NFV is for flexible architecture, MANO is for dynamic configuration changes, and MEC for IT processing.

Works addressed in [13] and [14] show an IoT architecture based on software-defined network (SDN). As in the previous works, they do not provide a single solution where the IoT operations are managed on the SDN platform. In [15], an SDN solution for IoT built with an open network operating system (ONOS) as an SDN orchestrator is proposed. The architecture is suitable for orchestrating end-to-end service chains deployed across heterogeneous SDN/NFV domains and define a related high-level and vendor-agnostic intent-based northbound interface (NBI).

In this article, we first propose an edge-based IoT platform for AI (EBI-PAI) by integrating serverless [16]–[18] and SDN controller into edge computing. With EBI-PAI, users can utilize unified interfaces, e.g., URLs, to access the distributed edge computing resources. Such that users do not need to face the complexity of underlying infrastructures. Their access requests will be redirected to an appropriate data center after domain name server (DNS) mapping. The selected edge data center should cover the user geographically and satisfy its QoE requirements at the same time. The redirection strategies are determined by an SDN controller, which monitors the whole network and installs the computed mapping policies into the DNS server. The mapping strategies will be optimized to satisfy the QoE requirements of users.

The deployment of MEC is incremental because the micro data centers bring additional costs, including server purchases, installation, and deployment, and the service providers have a limited budget. Thus, the whole network cannot be upgraded for one night. Besides, multiple base stations (BSs) can share an MEC data center, and it is not necessary to equip every BS in remote areas with separate MEC data centers. How to deploy MEC data centers with a limited budget is an important issue for service providers.

In this article, we formulate the problem and find the optimal deployment scheme, which can save costs while satisfying the QoE requirements. The solution to the problem is not straightforward, because different locations incur different costs and user experiences, including delay, bandwidth, etc., and the server load and network condition are changing dynamically. We prove that the problem is NP-complete and propose two heuristic algorithms to solve the problem with polynomial time.

We implement the prototype of EBI-PAI based on Openwhisk [19], which is an Apache opensource project for serverless. We deploy it on a testbed with five servers. We conduct comprehensive simulations using both generated and real-world topologies. We also evaluate the proposed algorithms using the BS data set of Shanghai. Our simulation results show that the heuristic algorithms can achieve near-optimal results in 99% of experimental scenarios. EBI-PAI can increase QoE by about 10%–25% at a given cost budget with the generated topologies, and by 15%–35% with real-world topologies. We compare the proposed algorithms with the current de-facto, and the results show that they greatly improve the performance. We carry out a case study by deploying a real-time face recognition application in real networks. The results further validate the above results.

In this article, our contributions are as follows.

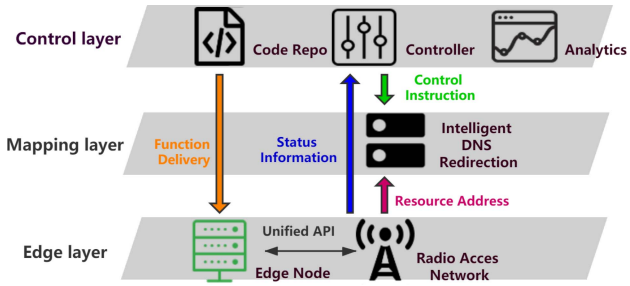


Fig. 1. System architecture of EBI-PAI.

1) We propose EBI-PAI, a QoE-aware edge computing platform based on the SDN controller and serverless computing technologies, to solve the problems of resource scheduling, service defining in future AI + IoT scenarios.
2) To save cost for service providers and satisfy QoE requirement for users, we study the QoE-aware server deployment problem and design heuristic algorithms with near-optimal performance.
3) We extensively simulate the proposed algorithms with both generated and real-world topologies, and a real-world BS data set collected over Shanghai, China. Evaluation results show that EBI-PAI increases QoE by 10%–35% with the same budget, compared with the current deployment scheme.

The remainder of this article is organized as follows. Section II introduces the framework of EBI-PAI. Section III formulates the deployment problem and proves the complexity of it. In Section IV, we propose heuristic algorithms to solve the problem with polynomial time. In Section V, we evaluate the performance of EBI-PAI. We show the prototype of our implementation in Section VI and conduct a case study with a real-time face recognition application in Section VII. We review related work in Section VIII and conclude our paper in Section IX.

II. EBI-PAI Design

The difference between our controller and SDN controller is that the SDN controller customizes the forwarding table of routers and switches to control the forwarding path, while the EBI-PAI controller manipulates the path in the application layer. The system architecture of EBI-PAI is demonstrated in Fig. 1.

A. Platform Controller

EBI-PAI sets up a centralized controller in the cloud based on SDN, which collects global network conditions and edge server resource information. Service requests from mobile devices are delivered to nearby BSs through one-hop wireless connections. All BSs upload information about current service requests, uplink and downlink bandwidth, and resource usage of the MEC server to the EBI-PAI controller through an SDN protocol [20], [21]. Through this real-time information, the EBI-PAI controller intelligently makes online assessment and prediction of the congestion in an edge network and a

load of each edge server, and the optimal resource allocation and scheduling strategy are obtained through predefined algorithms.

The controller also predicts the traffic distribution over networks and loads of MEC servers in the future. Traffic prediction is an important aspect of EBI-PAI, however, it is out of the scope of this article. Our work is orthogonal with the previous work on traffic prediction [22].

EBI-PAI is divided into three layers as follows.

1) *Control Layer:* The role of a centralized resource scheduling platform in edge computing is crucial. The distribution of functions can support multiple application scenarios on mobile devices. The centralized control eliminates the burden of directly configuring devices one by one in the edge network, which realizes dynamic customization of services in large-scale edge networks. It also accelerates the expansion of new application services in mobile-edge networks. First, the execution environment and code of the AI application are created by the developer and stored in the code repository. When an edge server receives a request from users, the controller delivers the computing environment to the edge servers. Second, based on real-time information, EBI-PAI predicts congestions in edge networks and loads of edge servers, computes a scheduling strategy, and sends instructions to each DNS server.
2) *Mapping Layer:* According to the instructions from the controller, the DNS server intelligently adjusts the forwarding path of the request in advance to balance the load of the edge server. By resolving local requests and forward users' requests to the selected edge server, subsequent requests from the user can be directly forwarded to the server.
3) *Edge Layer:* The edge computing platform adopts a serverless architecture, which is different from traditional virtual machines. Serverless architecture takes over the management of computing, storage, and network resources by establishing a service engine, shielding users from the underlying details. Users invoke the unified API interfaces without knowing the locations of services. Service execution is located in the edge server, which provides computing environments for different services. Service requests from mobile devices are called through Restful API. With DNS redirection, the request is forwarded to an edge server. The edge server is responsible for hosting the execution environment. The edge server also reports execution status, uplink and downlink bandwidth, and resource usage of the server to the controller.

B. Edge Computing Platform

By combining the idea of SDN and serverless technology, life cycles of edge resources and applications can be easily managed. This makes it possible to dynamically customize personalized services in a large-scale edge network and accelerate the configuration process of new application services.

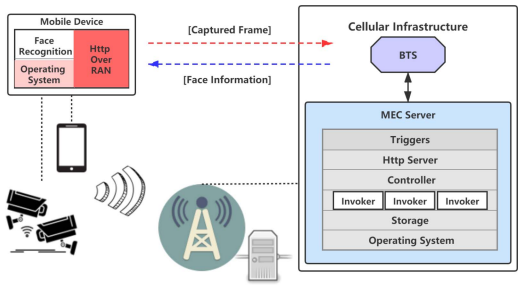


Fig. 2. EBI-PAI's serverless edge computing platform.

Fig. 2 shows the edge computing platform deployed on an MEC server based on the serverless technology (such as OpenWhisk). The entry point of the platform is a trigger associated with a specific event. In a face recognition application, for example, events that trigger functions include uploading images or using video frames captured by a device camera. These triggers make requests to the Http server, which exposes a Restful API in the form of callable functions.

To achieve network transparency and rapid redirection, a local DNS is deployed on the cellular infrastructure. Although DNS will introduce extra delay: 1) the delay could be reduced with a local DNS and 2) users only need to access DNS when first call the APIs and resolve them to IP addresses, and the subsequent accesses will be forwarded directly to MEC servers.

III. QoE-Aware Server Deployment

As the first step in the deployment of edge computing architecture, deployment of the edge server is the foundation and key [23]. Previous studies on edge server deployment have focused on algorithms for clustering access points with the target to find candidate locations for the servers as cluster centers. The QoE-aware server deployment problem discussed here differs from the traditional server deployment problem in two ways. First, we ensure a timed response of the access to every node in a network with the minimum number of servers deployed. Solutions for facility location, k-means clustering, and multiobjective constraint optimization all attempt to achieve the lowest facility opening costs, regardless of the bounded QoE requirements of each node. Second, we set the highest priority for responsiveness requirements of applications in edge computing.

Responsiveness, which refers to how fast the users can access the requested information, is an important type of QoE requirements needed by a wide range of applications. The desired level of performance can be specified in the form of service level agreements (SLAs) between the content/service providers and their customers, e.g., the response time of requests from domain A for Nasdaq.com home should not exceed 1 s; 95% of the requests from domain B for CNN.com home should complete in less than 2 s.

The access latency experienced by clients is affected by factors, including network latency (in terms of the round-trip time), server load, and network load. Similar to other studies [24]–[26], in this article, we mainly focus on the network

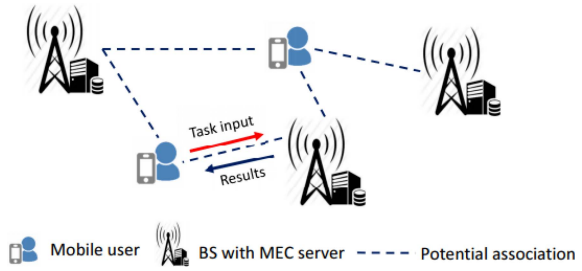


Fig. 3. Multicell, multiserver MEC system.

TABLE I
NOTATION TABLE

Variables	Meaning
V	Set of Base Stations (BS)
u, v, w	Base Station(BS) for server deployment
$s(v)$	Cost of placing server at BS v
$q(v)$	QoE requirement of BS v
$d(u, v)$	Distance between BS u and v
$c(u)$	Cover set of BS u
$r(u)$	Reverse cover set of BS u
x_i	Whether an edge server is placed at BS v_i

delay factor and use the communication cost involved in service request as a measure of responsiveness perceived by clients. The reason is that overloaded servers or bottleneck links can always be better provisioned to handle more user requests (e.g., by upgrading server clusters or purchasing more bandwidth). However, network latency cannot be improved by simply adding hardware resources. Since the communication cost from clients to the associated server receiving its request is independent of the deployment strategy, for simplicity, this part of the cost is not included in our analysis model.

A. Problem Formulation

We consider a multicell, multiserver MEC system as illustrated in Fig. 3, in which each BS is equipped with an MEC server to provide computation offloading services to the resource-constrained mobile users, such as smartphones, tablets, and wearable devices. In general, each MEC server can be either a physical server or a virtual machine with moderate computing capabilities provisioned by the network operator and can communicate with the mobile devices through wireless channels provided by the corresponding BS. Suppose we want to select appropriate locations among N BS cites distributed in a region to deploy edge servers. Given the edge servers, BS can offload the designated task to the neighboring edge server. We assume that each edge server is responsible for a subset of BS to process mobile user requests. We also assume that each edge server has unlimited computing resources to process mobile user requests, and each BS accessed the edge server directly, i.e., each BS is co-located with an edge server if provisioned.

Consider all BS in a region are connected to form a network represented by a graph $G = (V, E)$, where V is the set of BS and $E \subset V \times V$ is the set of physical or logical links between them. A weight $s(v)$ is associated with each BS $v \in V$, representing the cost of placing servers at v. Different sites may have different deployment costs. Moreover, a distance $d(u, v)$ is associated with each edge $(u, v) \in E$, representing the communication cost of forwarding requests for execution and associated response between u and v. We use "communication cost" as a term with a general sense, which can be different performance metrics, such as latency and hop count. We define $d(u, v)$ as the communication cost between u and v.

Services are invoked by the clients outside the network of servers. We assume each server receives requests from some group of clients (e.g., by statically configuring the clients, using DNS-based request direction, or intercepting requests in a transparent fashion [27]). If the service is hosted on the server receiving the request, the response is generated locally. Otherwise, the server forwards the request to some other server in the network for the first time and relays the response to the client.

We investigate the problem of choosing BSs in an MEC system to meet the QoE requirements of clients with the objective of minimizing the placement cost. The QoE requirements are specified in the form of a general distance metric. Every BS in the network has some QoE requirements on serving requests for its clients. The QoE requirement of each BS v is specified by an upper bound $q(v)$ on retrieval cost. If the response can be retrieved by v within a cost of $q(v)$, the QoE requirement is satisfied. Otherwise, the QoE requirement is violated. The QoE requirements associated with different BSs can be different. In general, a client would experience shorter access latency if a BS with server deployed is placed in its closer proximity. Therefore, it is desirable to allocate servers in the network closer to the clients with higher QoE requirements.

Definition 1: The QoE-aware server deployment problem in multicell, multiserver MEC system is to select a minimal subset of BS in a region, where all other BS's QoE requirement can be satisfied by this subset.

The objective of the QoE-aware server deployment problem is to find a deployment strategy that satisfies the QoE requirements of all BSs and involves the minimal cost. The corresponding integer programming problem can be formulated as follows:

$$\text{Minimize} \quad \sum_{i \in V} s_i x_i \tag{1}$$

$$\text{s.t.} \quad \sum_{d(v_i, v_j) \le q(v_i)} x_j \ge 1 \quad \forall i, j \in V \tag{2}$$

$$x_i \in 0, 1 \quad \forall i \in V. \tag{3}$$

The 0-1 variable x_i indicates whether an edge server is placed at BS v_i. Constraint (2) ensures all BS's QoE requirements are satisfied. The notations are listed in Table I.

B. NP-Completeness Results

We show that the minimum dominating set problem which is known to be a special instance of a minimum set-covering problem can be reduced to the QoE-aware server deployment problem.

Theorem 1: The QoE-aware server deployment problem is NP-complete.

Proof: We start with the definition of a dominating set. In the graph theory, if D is a subset of V and every vertex not in D is adjacent to at least one member of D, then subset D is said to be a dominating set of graph $G = (V, E)$. The dominating number $y(G)$ is the number of vertices in the minimum dominating set of G. The minimum dominating set problem involves finding a subset D with the minimum dominating number. It is a classical NP-complete decision problem in the computational complexity theory. Therefore, it is believed that there is no effective algorithm that finds all the minimum dominating sets of a graph. Then, let us define formally the solution of the QoE-aware server deployment problem, QAP as follows:

$$\left\{G = (V, E), d, \theta | \text{ There is } \widetilde{V} \leq V \text{ such that } d_i <= \theta\right\}.$$

Suppose that we are given $G = (V, E)$ as an instance of the minimum dominating set. We consider a delay d such that $d = 1$ for every $e \in E$, and deadlines $\theta = 1$ for each node $i \in V$. Suppose that $\widetilde{V}$ is a deployment set. Since the delay for each edge is 1 and the deadline is 1, a node must have a neighbor in $\widetilde{V}$. Therefore, $\widetilde{V}$ is also a minimum dominating set for G. On the other hand, if $\widetilde{V}$ is a minimum dominating set for G, then clearly, $\widetilde{V}$ is a deployment set for QAP with respect to the unit delay and the unit deadlines.

As mentioned above, the QoE-aware server deployment problem is NP-hard. Therefore, the amount of calculation required to obtain an accurate optimal solution is too computationally intensive to be useful in practice. To evaluate the performance of the heuristic algorithms proposed, we compute a lower bound on the cost of minimizing any feasible solution to the problem. We use a superoptimal algorithm based on the Lagrange relaxation to achieve this. As the name suggests, this algorithm may produce a better solution than the optimal solution because it may not be feasible. Nevertheless, it serves as a useful data point for comparison.

The upper bound of the optimal solution of an integer programming can be given by the optimal solution of the corresponding relaxed linear programming [28]. In our experiment, by replacing the last constraint $x_i \in 0, 1$ to $0 \leq x_i \leq 1$, we relax the integer programming to a regular linear programming and calculate the optimal solution to the latter. Although the solution to linear programming may not be integral (that is, not feasible in practice), it provides an upper bound on the optimal solution of the edge server deployment problem. ■

IV. Heuristic Algorithms for QoE-Aware Server Deployment

In this section, we present two heuristic algorithms for server deployment, all of which share the greedy approach.

A. Greedy Minimum Dominating Set Algorithm

The greedy minimum dominating set algorithm (GDSA) (as shown in Algorithm 1) starts with an empty deployment strategy $R = \emptyset$. It intends to insert one BS into R each iteration. At each step, the insertion alternative with the maximum heuristic function value is selected, which can also be called normalized benefit. The normalized benefit is defined as the increased number of QoE satisfied nodes divided by the increased provision cost. The procedure repeats until all BS's QoE requirements are satisfied.

Algorithm 1: GDSA

```
Input: G = (V, E), every node's QoE requirement q(i)
Output: Feasible deployment strategy
Calculate all-pairs shortest path distance;
Set Selected is initially empty;
Set NSelected include all vertices;
Set NCovered include all vertices;
/* Builds cover set */
for i = 1; i ≤ |V|; i++ do
    for j = 1; j ≤ |V|; j++ do
        if d(i, j) ≤ q(i) then
            add j to v_i's cover set;
while there exists unsatisfied vertices do
    /* Initialize maximum normalized benefit */
    Max_N = −1;
    for each v_i in NSelected do
        N_i = normalized benefit of the chosen vertex;
        if N_i > Max_N then
            Max_N = N_i;
            best_server = v_i;
    mark best_server is replicated;
    remove vertices satisfied by best_server from
    NCovered;
```

We analyze the time complexity of GDSA. First, GDSA calculates the shortest path between any two vertices in the graph. The classic Floyd algorithm is used in this step, whose time complexity is $O(|V|^3)$. Second, the algorithm traverses all vertices in the graph and takes vertices within the coverage as the cover set of each vertex. Since the cover set of a vertex may include all vertices in the graph, the time spent in this step is $O(|V|^2)$. In the third step, the vertex with the maximum normalized benefit is selected. Finally, repeat the third step until all the QoE requirements are satisfied. To sum up, the time complexity of GDSA is $O(|V|^2 + |V|^3) = O(|V|^3)$.

B. Greedy Cover for QoE-Aware

In this section, we propose a new heuristic algorithm called a greedy cover for QoE-aware (GCQA), which finds a good solution for the problem of QoE-aware server deployment in general graphs.

We start with the definition of the cover set and reverse cover set.

Definition 2: The cover set $c(u)$ of BS u is the set of BS that u can satisfy their QoE requirements, while the reverse cover set $r(u)$ of BS u is the set of BS that can satisfy the QoE requirement $q(u)$ of u.

Each BS has its own cover set and reverse cover set. If there is a server deployed on BS $w \in r(u)$, then BS w can satisfy user requests from u. Therefore, each BS in $r(u)$ is a candidate that can place the server to satisfy BS u. We first observe

Algorithm 2: GCQA

```
Input: G = (V, E), every node's QoE requirement q(i)
Output: Feasible deployment strategy
Find all-pairs shortest path distance;
/* Builds cover set and rever cover set */
for i = 1; i ≤ |V|; i++ do
    for j = 1; j ≤ |V|; j++ do
        if d(i, j) ≤ q(j) then
            add j to v_i's cover set;
        if d(i, j) ≤ q(i) then
            add j to v_i's reverse cover set;
Remove super reverse cover sets;
while there exists unsatisfied BS do
    select min_cover_set from reverse cover sets;
    /* Initialize maximum normalized benefit */
    Max_N = −1;
    for each v_i in min_cover_set do
        N_i = normalized benefit of the newly placed
        server;
        if N_i > Max_N then
            Max_N = N_i;
            best_server = v_i;
    mark best_server is placed;
    remove BS satisfied by best_server;
```

that if $r(u) \subset r(v)$, then $r(v)$ may need not be considered when choosing deployment sites. If we place a server on BS $w \in r(u)$, then BS w can satisfy both u and v.

Then, we observe that if $|r(v)| > |r(u)|$, BS v is easier to be satisfied than u. The reason is that if the reverse cover set $r(v)$ contains more elements, $r(v)$ is more likely to overlap with other reverse cover sets, so v has more opportunities to be satisfied when processing other reverse cover sets. Our intuition is that when selecting candidate BS to place server, the BS with the smallest reverse cover set is satisfied first, then the BS is also likely to satisfy the other BS with a larger reverse cover set. In this way, we may find a deployment strategy with fewer edge servers and reduced opening costs. Based on these observations and intuitions, we propose the GCQA algorithm as shown in Algorithm 2.

The first step in GCQA is to find the cover set and reverse cover set of each BS in the network. Second, we remove all super reverse cover sets $r(v)$ that contain some other reverse cover set, $r(u)$. That is, if $r(u) \subseteq r(v), u \neq v$, we remove v from those BSs that must be satisfied. In each subsequent step, GCQA chooses the smallest cover set c, examines every server s, and puts a replica on a server s in c with the highest normalized benefit.

Theorem 2: The GCQA algorithm has an asymptotic upper bound of $O(|V|^3)$ on the worst-case running time.

Proof: We analyze the time complexity of the three phases of GCQA. In the first stage, GCQA builds a cover set and reverse cover set for each BS in the network. The cover sets of each BS can be found by sequentially determining other $|V|$ BS in the network, and the cover sets of a BS also includes the BS itself. Since each BS has cover sets, it takes $O(|V|^2)$ build cover sets for all BS in the network. In the second stage, GCQA identifies and deletes all super reverse cover set in the network. In order to identify all super reverse cover sets, GCQA needs to check all $O(|V|^2)$ possible reverse cover set pairs. Checking a pair of reverse cover sets requires $O(|V|)$, so identifying and deleting all super cover sets takes the time of algorithm $O(|V|^3)$. In the final stage, the algorithm inserts servers into the network in turn until all BSs are satisfied. First, GCQA chooses the smallest reverse cover set, which can be done by sorting the reverse cover set by size. After finding the smallest reverse cover set c, GCQA calculates the normalized benefit of all BS in c and places the server on the BS with the largest normalized benefit. Due to the newly placed server, it takes $O(|V|)$ to calculate the increased number of satisfied BS and the increased cost, so it takes $O(|V|)$ to calculate the standardized benefits of the BS. The size of the cover set is $O(|V|)$ in the worst case, so we need to consider the cover set of $O(|V|)$. As a result, it takes $O(|V|\log|V| + |V|^3) = O(|V|^3)$ time to complete the last phase. ■

V. Implementation

Due to the ongoing evolution of industrial networks from Fieldbus technologies to Ethernet, a new opportunity has emerged to harness the benefits of SDN. SDN was first designed to orchestrate IT networks, but nowadays, some SDN controllers include plugins to connect in the southbound with IoT devices and networks [29], [30]. For example, the proposed platform can use OpenDaylight (ODL) [31], an SDN controller with a dedicated IoT plugin [32].

We use five high-performance Aliyun elastic compute service (ECS) instances to build the platform controller of EBI-PAI. Five instances run all core components of the controller. The configuration of each instance is 4-Core CPU, 8-GB memory, and 50 Mb/s bandwidth. Three edge computing clusters are deployed at the edge of the network and communicate with the platform controller through a control protocol. Each MEC server in the edge computing cluster runs the Ubuntu-14.04.1-server-amd64 operating system and employs Apache OpenWhisk to virtualize server resources to support serverless computing. Resource orchestration and management of the entire computing cluster are through Kubernetes.

Apache OpenWhisk [19] is an opensource serverless architecture implementation. The serverless architecture is a refined cloud computing model to process requested functionality without preallocating any computing capability. Provider-managed containers are used to execute functions (called actions), which are event triggered and ephemeral.

The core components of EBI-PAI's platform controller is depicted in Fig. 4. The controller is responsible for code distribution, status information collection, and user access control, and is composed of three modules: 1) access management; 2) distributor; and 3) platform gateway. The access management module provides an interface for the front-end page to operate the database, writes system events (such as user registration) to the message queue RabbitMQ, and provides a query function for statistics by initiating a request to time-series database Prometheus. The distributor delivers

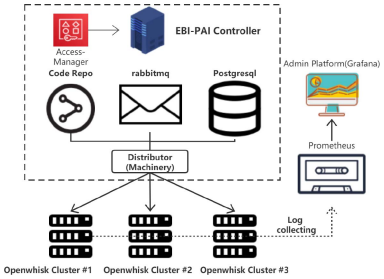


Fig. 4. Core components of EBI-PAI's platform controller.

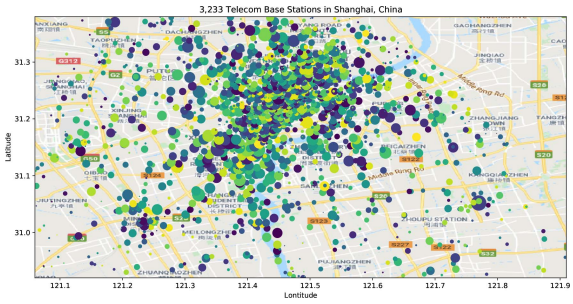


Fig. 5. Shanghai Telecom BS distribution.

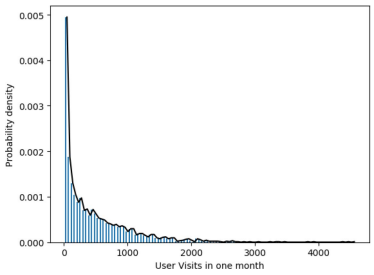


Fig. 6. User visits distribution on 3233 BSs.

TABLE II
INFORMATION OF A PART OF BSS

ID	Latitude	Longitude	User Number
7	31.2377	121.3825	5
16	31.4837	121.2748	165
193	31.0154	121.1548	367
446	31.2815	121.558	212
593	31.3068	121.5196	164
677	31.3011	121.1778	286
753	31.4534	121.2564	202
833	31.2567	121.4387	10
1005	31.3971	121.5077	24
1141	31.1860	121.4487	17
1325	31.2479	121.5134	53

application images and user access control information to edge computing nodes. The platform gateway collects function call information reported by the edge platform and provides user access control.

VI. SIMULATION

A. Simulation Setup

To evaluate the performance of the algorithms proposed in this article and compare it with other algorithms in the existing work, we simulate the behavior of the algorithms on a variety of network topologies and real-world data set. In this section, we discuss the network topologies and data set that we use in our evaluations. We then describe the performance metric that we use as a basis for comparing the algorithms.

1) Network Topologies and Data Set (Waxman Model): We choose to use Waxman model [33], which is the most commonly used model in previous works [34], [35]. In the experiments, we first use the Waxman model to randomly generate the topology of the network. The cost of each link is given by the Euclidean distance between the two endpoints. The topology generated by this method is not guaranteed to be single-connected. Since there are no isolated nodes or connected components in the actual network topology, it may take multiple runs to get a connected graph, and each time, we use the minimum spanning tree algorithm [36] to test the generated topology. The information of BSs is listed in Table II.

AT&T North America: This data set is provided by The Internet Topology Zoo website [37] and represents the backbone IP network built by AT&T between major U.S. cities. The entire network connects 25 major U.S. cities, with a total of 57 edges. The data set contains geographic location information for these 25 cities.

Telecom Data Set: The data set, provided by Shanghai Telecom [38], contains more than 7.2 million records of accessing the Internet through 3233 BSs from 9481 mobile phones for six months. For example, Fig. 5 shows the distribution of BSs. Each circle in the figure represents a BS in Shanghai, China. We use the size of circles to represent the user traffic on each BS. Fig. 6 shows the distribution of user visits for all BSs. It can be seen that the user visits have a clear power-law distribution. Approximately, 10% of BSs have more than 1000 user visits, while BSs with less than 200 user visits exceed 66%.

2) Alternative Deployment Approaches: To evaluate our algorithm on provisioning edges with bounded QoE requirements, we compare it with two alternative approaches in the existing work for server deployment. These approaches are described as follows.

Multirounds K-Medoids: We adapt the k-medoids clustering algorithm used in [39] to solve the QoE-aware server deployment problem. The tailored algorithm is called multiround K-medoids (MRKMs). First, MRKM sets K as the number of groups in the first iteration of the algorithm, then uses the k-medoids clustering algorithm to divide all BSs into K clusters, and deploys K edge servers on the central BSs of each cluster. As the center of each cluster, the sum of square distances between it and all other BSs in the cluster should be the smallest. Different from the common k-medoids algorithm, MRKM then checks whether the QoE requirements of each BS are met under the current deployment strategy after each iteration, that is, the distance from any BS to the central BS of its cluster is within the QoE requirements. If all of them are satisfied, the final server deployment strategy will be output; otherwise, the number of k-medoids clustering groups is increased by one, and MRKM enters the next round of iteration.

Multirounds Top-K (MRTK): We adapt the top-K greedy algorithm to solve the QoE-aware server deployment problem. The improved algorithm is called MRTK. In MRTK, all candidate BSs are arranged in descending order according to the number of mobile user requests received. In each iteration, the BS with the largest number of user requests in the unselected set is selected to place edge servers. Different from the top-K algorithm, MRTK checks whether the QoE requirements of each server are met under the current server deployment strategy after each iteration. If they are met, the final server deployment strategy will be output; otherwise, the next iteration will be carried out.

3) Performance Metric:

1) *Normalized Deployment Cost:* To compare the performance of the algorithms on the various network topologies and telecom data set, we use the relative performance of the algorithms as a metric. We define the relative performance as the ratio between the cost of the feasible solution found by the algorithm to the cost determined by the superoptimal algorithm. The relative performance is an appropriate metric since it reflects the cost we want to minimize. The smaller the value of the relative performance, the better the algorithm performs. The relative performance of 1 implies the algorithm finds an optimal solution, but the optimal solution need not necessarily have a relative performance of 1, since the superoptimal solution may not be achievable.
2) *Number of Servers Placed:* Under a certain network scale, node QoE requirements, and workload level, the number of servers that need to be placed in the network is obtained by the algorithm.
3) *Maximum Unsatisfied QoE:* Maximum unsatisfied QoE refers to the maximum distance between all unsatisfied nodes in the entire network with the nearest node on which the server has been placed. This value describes the capability of the algorithm to meet QoE requirements under given cost budget constraints.

B. Experimental Results

In simulation experiments, we use a total of three network topologies to evaluate the performance of the four algorithms mentioned above. These are the random network topologies generated using the Waxman model, the AT&T North America backbone IP network topology (AT&T North America), and the Shanghai Telecom BS data set (Telecom Data set). Among them, the edges in the first two networks are wired connections, and the entire network can be abstracted into a graph structure, while the vertices in the last network are telecom BSs scattered throughout the city, where the BSs communicate wirelessly.

1) Evaluation Results on the Waxman Model: We choose four important parameters into consideration, including network scales, node out-degrees, QoE tightness (delay constraint), and cost budget, which were commonly selected in previous works [40], [41]. The network scale is an important factor to consider. The placement cost obtained and algorithm efficiency can vary depending on the input number of nodes. Each simulation was performed ten times with the network topologies generated by the Waxman model with the values of parameters listed in Table III, and the normalized deployment costs are averaged. The distribution of node out-degrees obeys the power-law distribution. A simple way to generate a graph that obeys a power law is to use that power law to guide the construction of the graph, and node out-degree is a straightforward property to manipulate. The bigger the value of QoE, the more possible the QoE of the node is satisfied. We consider the influence of QoE tightness on various algorithms.

TABLE III
MAJOR PARAMETERS OF NETWORK SCALE SIMULATION

Parameter	Value
s in Waxman model	1000
α in Waxman model	0.15
β in Waxman model	0.75
QoE quantified by latency	100
Server placement cost	100
α in power-law distribution	2
β in power-law distribution	10

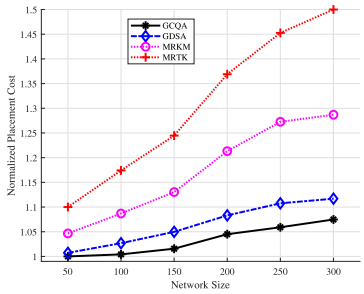


Fig. 7. Comparison for the relative performance of four algorithms in different network scales.

As expected, GCQA generally outperforms the other three. These performance trends are consistent over different settings of network parameters and network sizes examined.

Fig. 7 shows the relative performance of the four algorithms under the same QoE requirements and different scale network topologies as the network size increases compared to optimal. The horizontal axis represents the scale of the generated network. We change the number of initialized vertices to simulate networks of different sizes. The vertical axis represents the performance of the algorithm relative to the optimal. The smaller the value, the closer the result of the algorithm is to the optimal solution. We can see that the performance of the GCQA algorithm on networks of different sizes is the best among the four algorithms, and the gap between the GCQA algorithm and the optimal is not more than 10% in all cases; the performance of GDSA is second only to GCQA, and the gap with the optimal is kept within 15%. As the scale of the network increases, the performance of four algorithms decreases compared to the optimal, ranging from 10% to 40%.

Fig. 8 shows the relative performance of the four algorithms on the same network topology with the same scale and node out-degree as the QoE requirements of the nodes in the network change compared to optimal. The horizontal axis represents the QoE requirements of nodes. The smaller the value, the higher user's requirements for the delay, and the

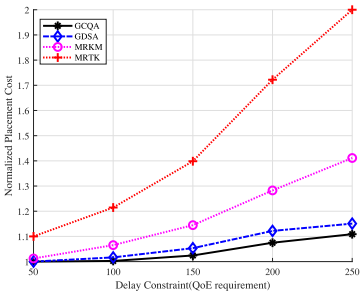


Fig. 8. Given the delay constraint, the normalized deployment cost of four algorithms.

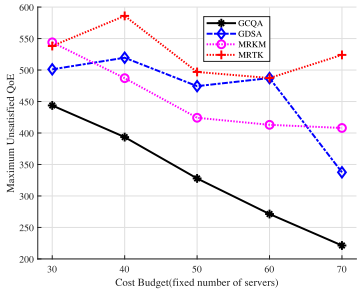


Fig. 10. Given cost budgets, the maximum unsatisfied QoE requirements of four algorithms.

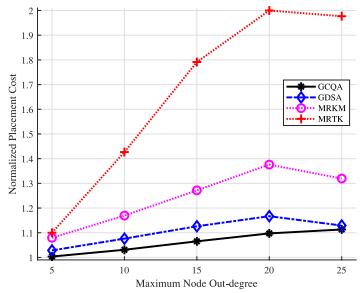


Fig. 9. Comparison for the relative performance of four algorithms in different node out-degrees.

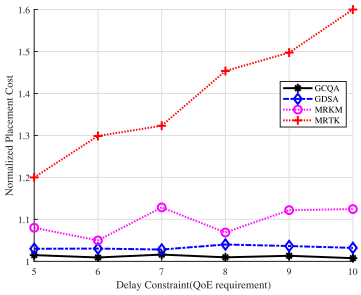


Fig. 11. Given the delay constraint, the comparison for the normalized deployment cost on the realistic topology.

more servers that need to be placed in the network accordingly. The vertical axis represents the performance of the algorithm relative to optimal. We can see that the performance of GCQA under different QoE requirements is the best among the four algorithms, and the gap between GCQA and the optimal does not exceed 10% in all cases; the performance of GDSA is second only to GCQA, and the gap with the optimal is kept within 20%. With the increase of QoE requirements, the performance of the four algorithms is compared with the optimal algorithm has decreased to a certain extent, ranging between 10% and 100%.

Fig. 9 shows how the four algorithms perform relative to the optimal on multiple network topologies with the same network scale and QoE requirements as the maximum out-degree of each node in the network changes. The horizontal axis represents the maximum out-degree of each node, and the larger the value, the more connections each node can establish with other nodes. The vertical axis represents the performance of the algorithm relative to the optimal. The results maintain the same trend as before. The performance of GCQA in different out-degrees is the best among the four algorithms, and the gap between GCQA and the optimal does not exceed 12% in all cases. The performance of GDSA is slightly inferior to GCQA, and the gap remains within 6%. With the increase of node out-degree, the performance of four algorithms compared with the optimal declines at the beginning and then increases. The turning point appears when the node out-degree is 20.

Fig. 10 shows the maximum unsatisfied QoE requirements of the four algorithms on the same network topology where the network size and QoE requirements are kept the same, under a given cost budget. The horizontal axis represents fixed deployment cost, represented by the number of servers placed in the network. The overall downward trend of the four curves in the figure shows that the greater the number of servers under the same QoE requirement, the fewer the nodes with unsatisfied demand, and the smaller the distance between the unmet nodes and the nearest node that has placed the server. The value of the vertical axis is selected from the maximum distance between all unsatisfied nodes in the entire network and the nearest node on which the server has been placed. The smaller the value, the better the algorithm's performance. We can see that GCQA's capability to satisfy QoE requirements is the best among four algorithms in different cost budgets. The downward trend of GCQA is obvious, which conveys that for newly added servers, GCQA can more effectively deploy servers in key locations to meet the node's QoE requirements to the greatest extent.

2) Evaluation Results on AT&T North America: Fig. 11 shows the performance of the four algorithms on the AT&T North America IP backbone network as the QoE requirements of the nodes in the network change. The horizontal axis represents the QoE requirements of nodes. We can see that when the network size is small, the performance of GCQA and GDSA under different QoE requirements is close, and the performance relative to the optimal is stable; and the gap between MRKM and the optimal is not more than 12%. With the increase of QoE requirements, the performance of MRTK has greatly decreased compared to the optimal.

Fig. 12 shows the largest unsatisfied QoE requirements of the four algorithms on AT&T North America network under a given cost budget. The value on the horizontal axis indicates the number of servers placed. The value is small because the network scale is also small. The four curves in the figure still show an overall downward trend. Different from the previous results in the generated topology, GCQA falls behind to GDSA for the first time, but as the number of placed servers increased,

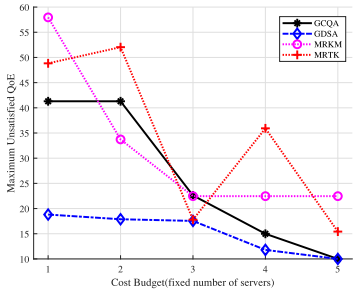


Fig. 12. Given cost budgets, the comparison for the maximum unsatisfied QoE requirements on the realistic topology.

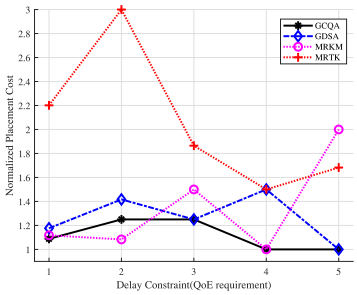


Fig. 13. Given the delay constraint, the normalized deployment cost of four algorithms.

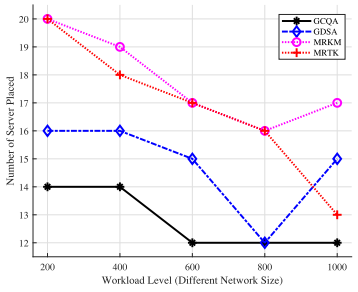


Fig. 14. Given the level of user connected (from 200 to 1000 over a period), comparison between the deployment costs of four algorithms.

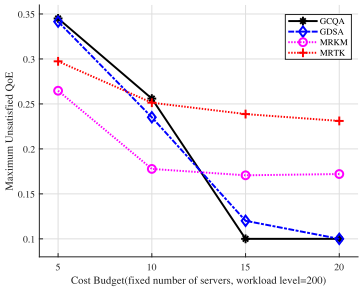


Fig. 15. Given cost budgets, the comparison for the maximum unsatisfied QoE requirements of four algorithms under a high workload level.

the performance of GCQA showed a trend of overtaking. We speculate that the reason for this phenomenon is that GCQA's server selection strategy is not as effective as the greedy GDSA algorithm in the initial stage, but as the number of servers to be deployed increases, GCQA's strategy is still optimal among all algorithms.

3) Evaluation Results on Telecom Data Set: Fig. 13 shows the relative performance of the four algorithms on the Shanghai Telecom BS data set as the QoE requirement of nodes in the network changes. The value on the horizontal axis is smaller than the previously generated network. The reason is that BSs are densely distributed in the city, and BSs are often deployed within a visible range. The communication delay between BSs and the user is typically a few milliseconds. GDSA and MRKM have outperformed and approached GCQA in a few cases, but the overall performance of GCQA is more stable, and the gap with the optimal remains within 23%, while the performance of GDSA has great fluctuations relative to the optimal, with a range of 5%–100%.

Fig. 14 shows the number of servers placed by the four algorithms on the Shanghai Telecom BS data set under the same QoE requirements as the network size changes. The horizontal axis represents the selection criteria of BS, and the vertical axis represents the number of servers that need to be deployed to meet QoE requirements. We select a subset of all BSs through user requests processed by BS within a cycle, which is the workload of BS. We evaluate the performance of four algorithms at five different workload levels and select a part of all BSs with workload above 200, 400, 600, 800, and 1000. The number of BSs in each set, which is also equal to the size of the network, is 1198, 869, 619, 443, and 309. The reason we do this is that in the early deployment of an MEC server, edge computing needs of areas with higher user density should be met first, and the load of BS is a good indicator that can be referred to. It can be seen that GCQA is still outstanding, with a maximum 25% reduction in deployment cost compared with GDSA, and the performance is more stable. On the whole, the four curves show a downward trend, because as the load level increases, the scale of the network decreases, and the number of servers to be deployed also decreases.

Fig. 15 shows the maximum unsatisfied QoE requirements of the four algorithms on the Shanghai Telecom BS network with the same network scale and QoE requirements under different server cost budgets. The value on the horizontal axis indicates the number of servers placed. The experimental results on larger network topology (node size exceeds 1000) confirm our previous speculation that GCQA's server selection strategy is not as effective as the other three algorithms in the initial stage, but with the number of servers to be deployed increased, the strategy of GCQA can quickly find the optimal solution. The correspondence in the figure is that after the number of placed servers exceeds 10, the largest unmet QoE requirement of GCQA remains unchanged, which is equal to the experimentally set QoE requirement, indicating that the algorithm has calculated the optimal deployment, under which all nodes' QoE requirements have been met.

Fig. 16 shows the largest unsatisfied QoE requirements of the four algorithms on the Shanghai Telecom BS network under the same network scale and QoE requirements under different server cost budgets. The difference from the previous experiment lies in the scale of the network. This time, a network consisting of BSs with a workload above 1000 was selected. The experimental results are basically the same as the previous experiment. The characteristic of GCQA has been verified again. Although the performance of the initial strategy

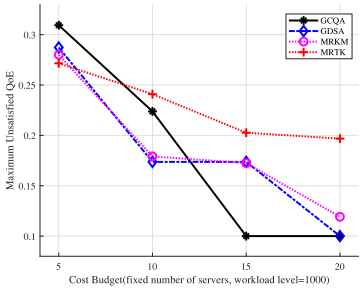


Fig. 16. Given cost budgets, the comparison for the maximum unsatisfied QoE requirements of four algorithms under a low workload level.

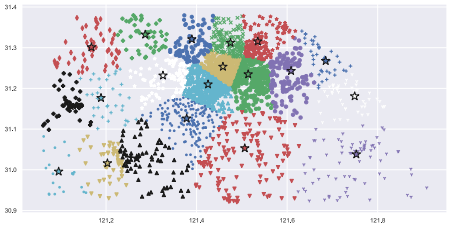


Fig. 17. K-medoids clustering of BS data set under $K = 20$.

is not as good as other algorithms, in the long run, this choice has a positive influence on subsequent performance.

Fig. 17 shows the processing results of the Shanghai Telecom BS data set using the K-medoids clustering algorithm. The experiment sets the number of clusters to 20, that is, all BSs are divided into 20 clusters according to distance. A server is deployed on the central BS of each cluster so that the sum of the distance between the BS and all other BSs in the cluster is the smallest, The maximum distance from the cluster center to other nodes in the cluster is used as the QoE requirement that the algorithm can satisfy.

C. Discussion

The GDSA algorithm is intuitive, while the GCQA algorithm solves the problem based on the concept of the reverse cover set. When the network size is large, GCQA obtained results closest to the optimal solution. But as shown in Figs. 13, 16, and 17, when the network size is small, e.g., the number of nodes is smaller than 10, the GDSA algorithm performs better than the GCQA.

VII. Real-Time Face Recognition for EBI-PAI: Case Study

We evaluate EBI-PAI with a real-world case study, which is a real-time face recognition application. In this case, we choose suitable sites to deploy edge servers.

In this case, we transfer data-intensive tasks of surveillance video streams from cloud to edge servers, to realize real-time video monitoring. There are two possible deployment approaches as shown in Fig. 18: 1) on the MEC server or 2) in a traditional cloud computing environment. The controller of EBI-PAI is implemented on a laptop. OpenWhisk is running on a virtual machine with 4-Core CPU, 4-GB memory, and 40-GB SSD. The cloud solution uses Aliyun and its face recognition services. The image is uploaded through

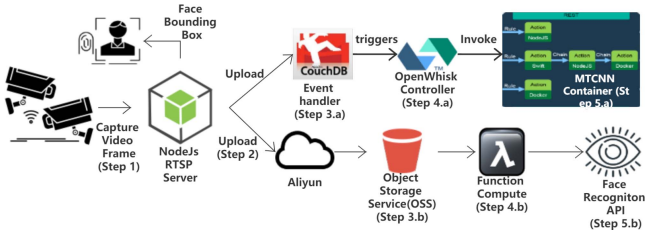


Fig. 18. Experimental setup for real-time face recognition.

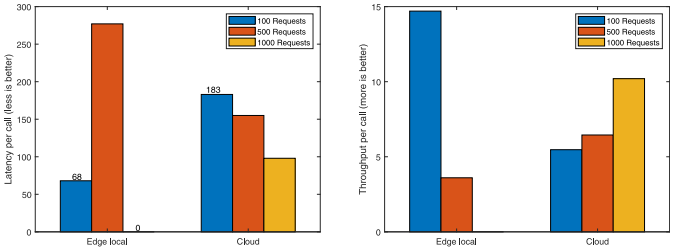


Fig. 19. Experimental results for 100, 500, and 1000 concurrent requests in the edge-based and cloud-based deployments. (a) Latency. (b) Throughput.

the Aliyun object storage service (OSS) bucket (step 3.b). An RTSP server based on Node.js captures and uploads images from devices such as cameras, provides endpoints for requests and image uploads (steps 1 and 2), and then triggers different subsequent steps based on two different deployments: steps 3.A, 4.A, and 5.A represent edge-based serverless solutions, steps 3.b, 4.b, and 5.b are for cloud-based solutions. In addition, the Node.js server collects metrics, such as latency, throughput, and computation time.

Fig. 18 shows the execution results of 100, 500, and 1000 concurrent requests of EBI-PAI and cloud-based solutions. We run each experiment ten times, calculate the average value, and show the results in Fig. 19(a). With 100 requests, the latency of EBI-PAI is 62% lower than the cloud solution. With 500 requests, the delay of EBI-PAI increases due to resource limitations. With 1000 requests, EBI-PAI fails to satisfy user demands. Fig. 19(b) shows the throughput under different configurations. With 100 requests, the throughput of EBI-PAI is nearly three times higher than the cloud-based solution. With 500 concurrent requests, the throughput of EBI-PAI decreases.

For deployment, we need to choose appropriate sites from 17 major cities to deploy edge servers. The goal of this deployment is to meet the QoE requirements of applications (delay should be less than 50 ms) while minimizing the deployment cost. The input includes a delay between major cities and the deployment cost of each city, and we compute the optimal deployment locations. Fig. 20 shows that MRKM needs to deploy six sites as there are six clusters with different shapes, GCQA needs to deploy five sites (the name is colored in red). We can see that GCQA saves 20% deployment cost compared with MRKM.

VIII. Related Work

The Internet is facing great challenges in the new era of IoT and AI, which will bring a huge amount of traffic [42]. Edge

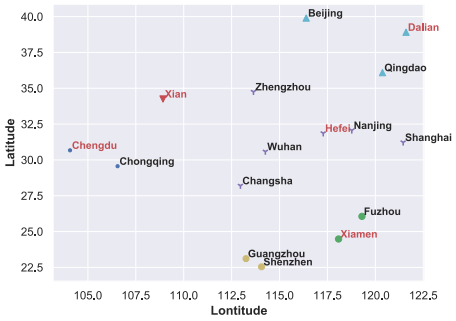


Fig. 20. Comparison of actual deployment effects of EBI-PAI under MRKM and GCQA algorithms.

computing improves user experiences by letting users access the computing resources with a lower distance. Nowadays, researchers have proposed many edge computing solutions in different fields [43], [44]. Different aspects of edge computing [45] have been taken into consideration, however, deployment of edge computing is still a big issue.

Edge servers, though densely distributed, have limited resources and require hosted applications and services to use them effectively. To address this challenge and the trend of various application scenarios, serverless paradigms for edge computing are emerging [16]–[18]. Serverless computing, a cloud computing model for stateless and event-driven applications, is expected to use temporary containers to remove the burden of always-on server infrastructures, further improving QoE. On the other hand, the short-term and mobility characteristics of edge users make serverless computing the most efficient and effective way to ensure resource utilization.

In edge computing, scheduling of edge computing resources remains challenging, and many resource scheduling schemes for edge computing have been proposed [46]–[48]. We note that traditional edge resource scheduling schemes focus on the optimization of propagation latency, load imbalance, etc., but ignores the deployment of distributed edge servers, which have a great impact on edge network performance [24]. In this article, we focus on the deployment of the edge servers.

The MEC server deployment problem was studied in [24], which shows that the locations of MEC servers have a great impact on the QoE and operational costs. Previous works usually use the clustering algorithm to find a suitable location to deploy servers among all candidate locations [23], [25], [26], [40], [49]–[51]. Among them, clustering algorithms use k-means [25], [49], graph theory [40], hierarchical tree-like structures [23], [26], multiobjective constraint optimization [50], and mixed-integer linear programming [51]. A heuristic decision-support management system for server deployment was proposed in [41].

However, guaranteeing QoE with traditional cluster-based algorithms is not satisfactory in terms of the cost of budgets. In this article, we develop EBI-PAI to optimize server deployment while satisfying QoE at the same time.

IX. Conclusion

In this article, we combined the idea of the software-defined and serverless computing paradigm, proposed an architecture that provides real-time services to users through MEC, and studied the implementation of the edge computing platform deployed within BSs, and edge server deployment issues. We proposed EBI-PAI, a lightweight resource scheduling platform, to solve resource scheduling, unified service access, and edge computing platform implementation in the future AI + IoT scenario. A new scheduling platform enables highly scalable, intelligent, and cost-effective use of edge resources with minimal configuration and setup. Fault tolerance is an important aspect of EBI-PAI, we will take it into consideration in our future work, e.g., introducing multiple controllers into the system to improve its reliability [52].

References

[1] Q. Rao and J. Frtunikj, "Deep learning for self-driving cars: Chances and challenges," in *Proc. IEEE/ACM 1st Int. Workshop Softw. Eng. AI Auton. Syst. (SEFAIAS)*, Gothenburg, Sweden, 2018, pp. 35–38.

[2] F. Civerchia *et al.*, "Remote control of a robot rover combining 5G, AI, and GPU image processing at the edge," in *Proc. Optical Fiber Commun. Conf. Exhibit. (OFC)*, San Diego, CA, USA, 2020, pp. 1–3.

[3] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: A computation offloading framework for smartphones," in *Proc. Int. Conf. Mobile Comput. Appl. Serv.*, 2010, pp. 59–79.

[4] P. K. Agyapong, M. Iwamura, D. Staehle, W. Kiess, and A. Benjebbour, "Design considerations for a 5G network architecture," *IEEE Commun. Mag.*, vol. 52, no. 11, pp. 65–75, Nov. 2014.

[5] Z. Shao *et al.*, "Real-time dynamic voltage loop scheduling for multi-core embedded systems," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 54, no. 5, pp. 445–449, May 2007.

[6] M. Qiu, E. H.-M. Sha, M. Liu, M. Lin, S. Hua, and L. T. Yang, "Energy minimization with loop fusion and multi-functional-unit scheduling for multidimensional DSP," *J. Parallel Distrib. Comput.*, vol. 68, no. 4, pp. 443–455, 2008.

[7] Y. Wu, H. Huang, C.-X. Wang, and Y. Pan, *5G-Enabled Internet of Things*. Boca Raton, FL, USA: CRC Press, 2019.

[8] H. Kaur, "A survey on 5G standardization for edge computing and Internet of Things," in *Proc. 15th Adv. Int. Conf. Telecommun.*, 2019, pp. 21–26.

[9] P. P. Ray, "A survey of IoT cloud platforms," *Future Comput. Informat. J.*, vol. 1, nos. 1–2, pp. 35–46, 2016.

[10] N. Matsuda, K. Takagi, S. Horiuchi, H. Aoki, and A. Akutagawa, "IoT network implemented with NFV," *NEC Techn. J.*, vol. 10, no. 3, pp. 35–40, 2016.

[11] N. Omnes, M. Bouillon, G. Fromentoux, and O. Le Grand, "A programmable and virtualized network & it infrastructure for the Internet of Things: How can NFV & SDN help for facing the upcoming challenges," in *Proc. 18th Int. Conf. Intell. Next Gener. Netw.*, Paris, France, 2015, pp. 64–69.

[12] N. Bizanis and F. A. Kuipers, "SDN and virtualization solutions for the Internet of Things: A survey," *IEEE Access*, vol. 4, no. 5591-5606, 2016.

[13] Y. Li, X. Su, J. Riekki, T. Kanter, and R. Rahmani, "A SDN-based architecture for horizontal Internet of Things services," in *Proc. IEEE Int. Conf. Commun.*, Kuala Lumpur, Malaysia, 2016, pp. 1–7.

[14] R. Vilalta *et al.*, "End-to-end SDN orchestration of IoT services using an SDN/NFV-enabled edge node," in *Proc. Optical Fiber Commun. Conf. Exhibit.*, Anaheim, CA, USA, 2016, pp. 1–3.

[15] W. Cerroni *et al.*, "Intent-based management and orchestration of heterogeneous OpenFlow/IoT SDN domains," in *Proc. IEEE Conf. Netw. Softw.*, Bologna, Italy, 2017, pp. 1–9.

[16] S. Nastic *et al.*, "A serverless real-time data analytics platform for edge computing," *IEEE Internet Comput.*, vol. 21, no. 4, pp. 64–71, Jul. 2017.

[17] L. Baresi, D. F. Mendonça, and M. Garriga, "Empowering low-latency applications through a serverless edge computing architecture," in *Proc. Eur. Conf. Serv. Orient. Cloud Comput.*, 2017, pp. 196–210.

[18] C. Cicconetti, M. Conti, and A. Passarella, "An architectural framework for serverless edge computing: Design and emulation tools," in *Proc. IEEE Int. Conf. Cloud Comput. Technol. Sci. (CloudCom)*, Nicosia, Cyprus, 2018, pp. 48–55.

[19] *Apache Openwhisk: An Open Source Serverless Cloud Platform*. Accessed: Jan. 5, 2020. [Online]. Available: https://openwhisk.apache.org/

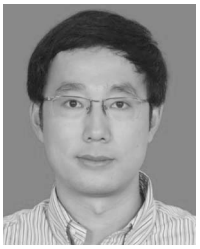
[20] *Opendaylight (ODL)*. Accessed: Jan. 17, 2020. [Online]. Available: https://www.opendaylight.org/
[21] *Open Network Operating SystEM (ONOS)*. Accessed: Dec. 23, 2019. [Online]. Available: https://www.opennetworking.org/onos/.
[22] J. Wu, Y. Peng, M. Song, M. Cui, and L. Zhang, "Link congestion prediction using machine learning for software-defined-network data plane," in *Proc. Int. Conf. Comput. Inf. Telecommun. Syst.*, Beijing, China, 2019, pp. 1–5.
[23] H. Sinky, B. Khalfi, B. Hamdaoui, and A. Rayes, "Adaptive edge-centric cloud content placement for responsive smart cities," *IEEE Netw.*, vol. 33, no. 3, pp. 177–183, May/Jun. 2019.
[24] Z. A. Qazi, P. K. Penumarthi, V. Sekar, V. Gopalakrishnan, K. Joshi, and S. R. Das, "KLEIN: A minimally disruptive design for an elastic cellular core," in *Proc. Symp. SDN Res.*, 2016, pp. 1–12.
[25] J. Liu, U. Paul, S. Troia, O. Falowo, and G. Maier, "K-means based spatial base station clustering for facility location problem in 5G," in *Proc. Southern Africa Telecommun. Netw. Appl. Conf. (SATNAC)*, 2018, pp. 406–409.
[26] J. Jiao, L. Chen, X. Hong, and J. Shi, "A heuristic algorithm for optimal facility placement in mobile edge networks," *KSII Trans. Internet Inf. Syst.*, vol. 11, no. 7, pp. 3329–3350, 2017.
[27] X. Sun and N. Ansari, "Latency aware workload offloading in the cloudlet network," *IEEE Commun. Lett.*, vol. 21, no. 7, pp. 1481–1484, Jul. 2017.
[28] V. V. Vazirani, *Approximation Algorithms*. Berlin, Germany: Springer-Verlag, 2003.
[29] *Cord (Central Office Re-Architected as A Datacenter) Platform*. Accessed: Dec. 16, 2019. [Online]. Available: https://www.opennetworking.org/cord/.
[30] *Opendaylight Controller Overview*. Accessed: Jan. 25, 2020. [Online]. Available: https://docs.opendaylight.org/en/stable-magnesium/user-guide/opendaylight-controller-overview.html
[31] *Opendaylight*. Accessed: Dec. 12, 2019. [Online]. Available: https://www.opendaylight.org/
[32] *IoTDM Developer Guide*. Accessed: Jan. 14, 2020. [Online]. Available: https://docs.opendaylight.org/en/stable-fluorine/developer-guide/iotdm-developer-guide.html
[33] B. M. Waxman, "Routing of multipoint connections," *IEEE J. Sel. Areas Commun.*, vol. 6, no. 9, pp. 1617–1622, Dec. 1988.
[34] M. Piechowiak and P. Zwierzykowski, "The evaluation of multicast routing algorithms with delay constraints in mesh networks," in *Proc. 8th Int. Symp. Commun. Syst. Netw. Digit. Signal Process. (CSNDSP)*, Poznan, Poland, 2012, pp. 1–5.
[35] A. Malik, B. Aziz, M. Adda, and C. Ke, "Optimisation methods for fast restoration of software-defined networks," *IEEE Access*, vol. 5, pp. 16111–16123, 2017.
[36] R. Sedgewick and K. Wayne, *Algorithms*. Upper Saddle River, NJ, USA: Addison-Wesley Prof., 2011.
[37] (2012). *AT&T North America*. [Online]. Available: http://www.topology-zoo.org/dataset.html.
[38] (2018). *The Telecom Dataset*. [Online]. Available: http://sguangwang.com/TelecomDataset.html
[39] L. Ma, J. Wu, and L. Chen, "DOTA: Delay bounded optimal cloudlet deployment and user association in WMAN," in *Proc. 17th IEEE/ACM Int. Symp. Clust. Cloud Grid Comput. (CCGRID)*, Madrid, Spain, 2017, pp. 196–203.
[40] F. Zeng, Y. Ren, X. Deng, and W. Li, "Cost-effective edge server placement in wireless metropolitan area networks," *Sensors*, vol. 19, no. 1, p. 32, 2019.
[41] H. Yin *et al.*, "Edge provisioning with flexible server placement," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 1031–1045, Apr. 2017.
[42] X. Cheng, Y. Wu, G. Min, A. Y. Zomaya, and X. Fang, "Safeguard network slicing in 5G: A learning augmented optimization approach," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 7, pp. 1600–1613, Jul. 2020, doi: 10.1109/JSAC.2020.2999696.
[43] S. Wang *et al.*, "Adaptive federated learning in resource constrained edge computing systems," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1205–1221, Jun. 2019.
[44] Z. Yan, J. Ge, Y. Wu, L. Li, and T. Li, "Automatic virtual network embedding: A deep reinforcement learning approach with graph convolutional networks," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1040–1057, Jun. 2020.
[45] L. Cui, S. Yang, Z. Chen, Y. Pan, Z. Ming, and M. Xu, "A decentralized and trusted edge computing platform for Internet of Things," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 3910–3922, May 2020.
[46] X. Li, J. Wan, H. Dai, M. Imran, M. Xia, and A. Celesti, "A hybrid computing solution and resource scheduling strategy for edge computing in smart manufacturing," *IEEE Trans. Ind. Informat.*, vol. 15, no. 7, pp. 4225–4234, Jul. 2019.
[47] J. Zhang, W. Xia, F. Yan, and L. Shen, "Joint computation offloading and resource allocation optimization in heterogeneous networks with mobile edge computing," *IEEE Access*, vol. 6, pp. 19324–19337, 2018.
[48] L. Cui *et al.*, "Joint optimization of energy consumption and latency in mobile edge computing for Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4791–4803, Jun. 2019.
[49] Y. Guo, S. Wang, A. Zhou, J. Xu, J. Yuan, and C.-H. Hsu, "User allocation-aware edge cloud placement in mobile edge computing," *Softw. Pract. Exp.*, vol. 50, no. 5, pp. 489–502, 2020.
[50] S. Wang, Y. Zhao, J. Xu, J. Yuan, and C.-H. Hsu, "Edge server placement in mobile edge computing," *J. Parallel Distrib. Comput.*, vol. 127, pp. 160–168, May 2019.
[51] M. Bouet and V. Conan, "Mobile edge computing resources optimization: A geo-clustering approach," *IEEE Trans. Netw. Serv. Manag.*, vol. 15, no. 2, pp. 787–796, Jun. 2018.
[52] S. Guo, S. Yang, Q. Li, and Y. Jiang, "Towards controller placement for robust software-defined networks," in *Proc. IEEE 34th Int. Perform. Comput. Commun. Conf. (IPCCC)*, Nanjing, China, 2015, pp. 1–8.

Shu Yang received the B.Sc. degree from the Beijing University of Posts and Telecommunications, Beijing, China, in 2009, and the Ph.D. degree from Tsinghua University, Beijing, in 2014.

He is currently an Associate Researcher with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China. His research interests include network architecture, edge computing, and high-performance router.

Kunkun Xu received the B.Sc. degree in e-commerce from Shanxi Normal University, Xi'an, China. He is currently pursuing the M.Sc. degree with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China.

His current field placement is with the National Engineering Laboratory for Big Data System Computing Technology. He is interested in edge computing, machine learning, and deep neural networks.



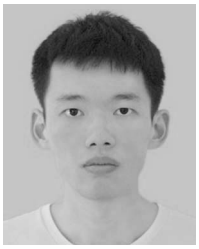
Laizhong Cui (Senior Member, IEEE) received the B.S. degree from Jilin University, Changchun, China, in 2007, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2012.

He is currently a Professor with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China. He is also with the Peng Cheng Laboratory, Shenzhen, China. He led more than ten scientific research projects, including the National Key Research and Development Plan of China, the National Natural Science Foundation of China, the Guangdong Natural Science Foundation of China, and Shenzhen Basic Research Plan. He has published more than 70 papers, including IEEE TRANSACTIONS ON MULTIMEDIA, the IEEE INTERNET OF THINGS, the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, the *ACM Transactions on Internet Technology*, the IEEE TRANSACTIONS ON COMPUTATIONAL BIOLOGY AND BIOINFORMATICS, and IEEE NETWORK. His research interests include future Internet architecture and protocols, edge computing, multimedia systems and applications, blockchain, Internet of Things, cloud and big data computing, software-defined network, social network, computational intelligence, and machine learning.

Prof. Cui serves as an Associate Editor or a member of Editorial Board for several international journals, including the *International Journal of Machine Learning and Cybernetics*, the *International Journal of Bio-Inspired Computation*, the *Ad-Hoc and Sensor Wireless Networks*, and the *Journal of Central South University*. He is a Senior Member of CCF.

Zhongxing Ming received the B.Eng. degree from the College of Software Engineering, Jilin University, Changchun, China, in 2009, and the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2015.

He is currently an Associate Research Fellow with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China. His research interests include future Internet architecture, Internet of Things, blockchain, and edge computing.



Ziteng Chen received the B.Sc. degree from Shenzhen University, Shenzhen, China, in 2018, where he is currently pursuing the M.Sc. degree.

His research interests include software-defined network, edge computing, and blockchain.

Zhong Ming (Member, IEEE) received the Ph.D. degree in computer science and technology from Sun Yat-Sen University, Guangzhou, China, in 2003.

He is currently a Professor with the National Engineering Laboratory for Big Data System Computing Technology and College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China. His research interests include software engineering and Web intelligence.