Reward-Oriented Task Offloading Under Limited Edge Server Power for Multiaccess Edge Computing

Minseok Song, *Member, IEEE*, Yeongju Lee, and Kyungmin Kim

***Abstract*—In multiaccess edge computing (MEC), tasks are offloaded from mobile devices to servers at the edge of the network. This speeds up task processing without incurring the latency required to reach central servers. However, the power used by edge servers is significant and needs to be cost-effective. We propose a scheme in which tasks are offloaded to servers with the aim of maximizing a reward within a limited power budget, server processing capacities, and wireless network coverage. Our algorithm determines the maximum utilization of each server while favoring the offloading of tasks with high ratios of reward to power requirement. We model the task allocation problem using a minimum-cost-maximum-flow graph, and propose two edge allocation algorithms, one of which is extended to allow task splitting, which offload tasks subject to server capacity by searching for the highest reward. In simulations, our scheme achieved between 8% and 80% higher rewards than alternative schemes, under the same power constraints.**

***Index Terms*—Edge server (ES) allocation, multiaccess edge computing (MEC), power capping, task offloading.**

I. Introduction

IMPROVEMENTS in network and system technologies have increased the feasibility of applications which rely on wireless networks [1]–[3]. Examples include augmented reality and smart traffic control systems based on connected Internet-of-Things (IoT) devices, including sensors, smart personal devices and driverless vehicles. The number of smart IoT devices is increasing rapidly: 500 billion smart IoT devices are predicted to be required in smart city applications by 2030 [2]. These applications will need to process the huge amounts of data that will be produced by these devices.

Some IoT devices must perform tasks requiring fast computation [4]–[7]. In addition, they typically involve periodic collection of sensor data that needs to be processed periodically [8], [9]. If their computational capability is limited, these tasks will sometimes need to be offloaded to cloud servers. But the delay imposed by the core network around the central servers in the cloud means that a cloud-based approach may not be suitable for IoT-based systems that require rapid processing [4]–[6].

Edge computing addresses this issue. Edge servers (ESs) located at the edges of cloud provide fast processing, as well as reducing the load on the central servers [1]–[3]. The demands for edge computing are expected to grow rapidly. Gartner [10] reports that by 2023, more than 50% of large enterprizes will have at least six IoT applications requiring edge computing. Edge computing services can be implemented by locating an ES at each base station (BS), or connecting an ES to several BSs. This is known as multiaccess edge computing (MEC) [1], [11], [12]. An IoT device located within the coverage of these BSs can then offload tasks to the ES at, or accessible through, that BS.

Edge infrastructure providers (EIPs) have now begun to provide edge-based services to customers [13], [14]. EIPs install ESs on the edges of a cloud and use them to provide computing power to their customers. Sharing an ES between customers improves resource utilization and obtains a financial return for the EIP. These financial benefits for EIP can be expressed as rewards and the contractual arrangements that an EIP has with its customers will determine the priorities operated by the servers [13], [14].

Although edge computing technology has effectively solved core network congestion and latency issues in current cloud computing systems, the available ESs may not have sufficient capacity to process all the tasks which IoT devices are requesting to offload [15], [16]. To utilize limited ES capacity more effectively, it is necessary to decide which tasks are offloaded within the context of a maximization of the total reward for the EIP [13], [17].

The energy efficiency of edge-based systems is a very significant issue for EIPs [18]–[20]. A typical ES is a microdatacenter at a remote location, and its power consumption can be expected to be of the order of 10 kW. Thousands of such servers will consume megawatts, which is a major concern for EIPs. In particular, power capping is essential to ensure that the power drawn by ESs always stays below a predefined power limit [21]. The power consumed by a server depends on its CPU utilization [22]–[26]. This is determined by the

Manuscript received August 20, 2020; revised November 18, 2020 and January 27, 2021; accepted February 28, 2021. Date of publication March 11, 2021; date of current version August 24, 2021. This work was supported in part by the Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT, under Grant 2017M3C4A7080248, and in part by the Basic Science Research Program through the NRF funded by the Ministry of Education under Grant NRF-2020R1F1A1068984. *(Corresponding author: Minseok Song.)*

Minseok Song and Kyungmin Kim are with the Department of Computer Engineering, Inha University, Incheon 22212, South Korea (e-mail: mssong@inha.ac.kr; kkmin00000@gmail.com).

Yeongju Lee is with System Integrated Group, FPT Software Korea, Seoul 07522, South Korea (e-mail: celesty0128@gmail.com).

Digital Object Identifier 10.1109/JIOT.2021.3065429

server's workload [28]–[30], which in turn depends on the number of offloaded tasks that an ES accepts. This tradeoff does not seem to have been addressed in previous work on ES management.

We propose an MEC architecture, which maximizes a reward, related to profit, accrued while limiting the power consumed by the ESs. We start by examining the power characteristics of the ESs and formulate an optimization problem that determines which ES should run each offloaded task and hence the maximum allowable CPU utilization of each ES. We address this problem with an algorithm which predicts the reward at each CPU utilization, taking into account the power consumption associated with that utilization. We then perform task allocation using a minimum-cost-maximum-flow (MCMF) graph, and propose two versions of a task allocation algorithm (one allows task splitting and the other does not) in which tasks associated with higher rewards are allocated first.

The remainder of this article is organized as follows. We review related work in Section II. We describe our system architecture and formulate the optimization problem in Section III, propose a CPU utilization determination algorithm in Section IV, and task allocation algorithms in Section V. We assess the effectiveness of these schemes in Section VI and draw conclusions in Section VII.

II. Related Work

Many edge computing techniques have been proposed to offload computation from IoT devices. Min *et al.* [4] used a reinforcement learning technique to determine the rate at which computations are offloaded from a mobile device on the basis of its current battery level and recent rate of radio transmission so as to make effective use of IoT battery capacity. Wang *et al.* [5] used a similar technique to address scenarios in which the mix of tasks to be offloaded is continually changing, by applying offloading patterns which are common to a number of different applications. Samanta *et al.* [6] presented task offloading schemes which aim to minimize total latency, including the delays due to task execution, queueing and offloading, while aiming to fulfill the Quality-of-Service (QoS) requirements of tasks. Zhao *et al.* [7] presented an analytical scheme that aims to maximize the probability that tasks satisfy their delay requirements when there is limited computational support from ESs. Sarrigiannis *et al.* [31] proposed a scheme that initially allocates virtual network functions (VFNs) to appropriate edge tiers by taking latency requirements, and then scales, migrates and destroys them to adapt network traffic. They [32] also presented a VFN lifecycle management scheme specifically for a fog-enabled cellular vehicle-to-everything architecture.

Sun *et al.* [33] presented task offloading schemes specifically for vehicles in which task delay and computing resource consumption are minimized. Ren *et al.* [34] considered cloud and edge computing resources collectively and proposed a task splitting scheme to minimize the latency incurred in offloading. Lai *et al.* [15] looked at the allocation of mobile devices to BSs in a way that maximizes QoS by formulating bin-packing problem based on the coverage of network cells. They also developed this scheme by applying a game-theoretic approach to the problem of edge device allocation and proposed a decentralized algorithm for its solution [16]. However, none of these works addresses the issue of power consumption by ESs.

Several authors have tackled the issue of ES placement in an MEC architecture. Wang *et al.* [11] studied the placement of ESs in smart cities to reduce the access delay experienced by mobile devices, while taking account of ES workloads. Lahderanta *et al.* [12] presented an algorithm for server placement which aims to minimize the distances between servers and their associated wireless access points (WAPs), while taking into account server capacity constraints. Xu *et al.* [35] introduced an ES placement scheme for 5G networks which balances workloads among ESs based on a decision tree. However, none of these schemes for server placement and workload balancing consider energy consumption.

Other schemes for ES deployment and allocation do consider commercial issues. For example, Anglano *et al.* [13] presented a scheme for assigning devices while maximizing overall profit in the context of revenues, infrastructure running costs and contractual penalties for QoS violations. Nguyen *et al.* [14] used a market equilibrium solution to promote the utilization of ESs in a way that maximizes revenue for EIPs. Hao *et al.* [17] presented a video caching and processing scheme specifically for multibit-rate video streaming to maximize profit. Xu *et al.* [36] presented an auction-based scheme for sharing computing resources among micro data centers, in a way that maximizes overall utility for EIPs. Riaz *et al.* [37] presented an algorithm that maps IoT devices to ESs with the aim of minimizing overall latency for a particular workload. Antonopoulos [38] suggested that the bankruptcy problem can be effectively used to allocate mobile devices to the ESs with limited resources. Samanta and Chang [39] presented an adaptive offloading scheme that maximizes total revenue while minimizing service latency by giving different priorities based on latency requirements using the Lagrange multiplier. Huang *et al.* [40] presented a task scheduling algorithm with the aim of maximizing total revenue by the EIP given the limited resource capacity of each ES. Again none of these schemes consider energy issues for ESs.

There has been a lot of work [29], [41]–[47] on reducing energy consumed by mobile devices using task offloading techniques. For example, You *et al.* [41] proposed an algorithm that minimizes the sum of weighted energy consumed by mobile devices under the constraints on the computational latency. Fu *et al.* [42] presented an algorithm that allows task offloading to ESs and harvesting energy from the BS to minimize the energy consumed by mobile devices, and Zhou *et al.* [43] proposed an energy-aware workload offloading scheme specifically for vehicular edge computing. Xiong *et al.* [47] proposed a UAV trajectory optimization scheme with the aim of minimizing energy required for task offloading from IoT devices to unmanned aerial vehicles.

There have been studies dealing with issues when power or edge resources are limited in MEC architectures as summarized in Table I. For example, several works presented schemes that turn off some of the ESs to address power limitation issues [19], [20], [48], [49], [50]. However, all of these energy-saving techniques completely turn off the ESs,

TABLE I
SUMMARY OF THE LITERATURE ON THE HANDLING OF POWER AND COMPUTATION BUDGET CONSTRAINTS UNDER MEC

Ref.	Scheme objectives	Solutions/techniques
[19]	Edge server deployment to minimize energy consumption subject to distance constraints between BSs and ESs	Algorithm for minimizing the number of active servers based on particle swarm optimization
[20]	Reducing the total energy consumption of edge servers	Turning off edge servers which are not processing workload
[48]	Minimizing the average delay cost subject to a long-term energy budget	Selective BS sleep determination algorithm based on Lyapunov optimization
[49]	Minimizing the number of active servers subject to users' stochastic delay distribution constraints	Distributed selective server activation algorithm based on the Theory of Minority Games
[50]	Minimizing energy consumption subject to server capacity budget in small-cell BSs	Dynamic server switching algorithm in which under-utilized edge servers are selectively turned off
[52]	Minimizing maximum task execution time under storage and computational budget constraints	Greedy heuristic algorithm based on the cost and time of each task execution
[53]	Minimizing response latency subject to cost constraints	Multi-slot service deployment algorithm based on Lyapunov optimization
[54]	Minimizing application execution delay subject to resource constraints	Edge resource rental algorithm based on Multi-armed Bandit optimization
[55]	Minimizing task execution time subject to task dependency constraints	Greedy offloading algorithm based on edge-cloud or edge-edge cooperation
[56]	Maximizing profit subject to edge server capacity and latency constraints	Deployment of network services to edge systems to match matching the network to service providers
[57]	Minimizing task execution delay subject to the energy constraints of users	User mobility management algorithm based on Lyapunov optimization
[58]	Minimizing delay costs subject to a long-term energy budget in small-cell base stations	Online algorithm to offload tasks to peer base stations based on Lyapunov optimization
[59]	Maximizing the total QoS of the system while managing the expected risk of exceeding the energy budget	Learning-based sample average approximation based on historical data

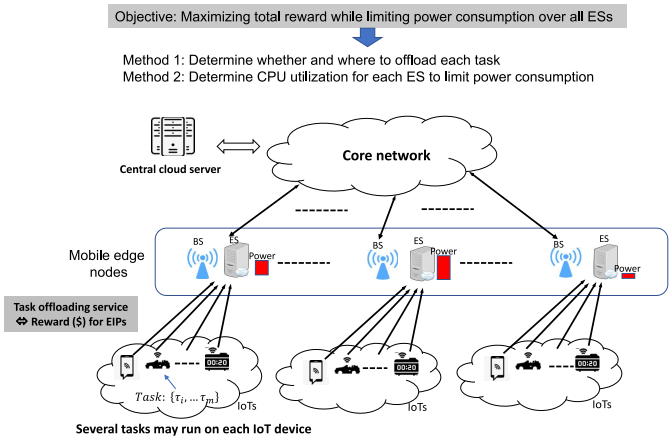

Fig. 1. MEC architecture.

complicating the network management of the MEC architecture [51]. For example, if the server cannot be reached while it is being turned off, then real-time reporting, software updates and control messages can be missed during that period [51].

Several authors have addressed the issues of reducing delays caused by selective task offloading due to the limitation of ES capacity [52]–[56] or energy budget [57]–[59]. Our scheme differs from these works in the following aspects. In terms of problem formulation, we use a flow network graph model to effectively represent the ES network coverage and the nonlinear relationship between CPU utilization and server power. Then, to accurately limit power consumption, we formulate the problem of controlling CPU utilization while making task offloading decisions. In terms of algorithm methodology, we first present an algorithm to limit power consumption based on the MCMF graph and propose two task offloading algorithms (one allows task splitting and the other does not) based on methods to find the path of minimum cost in the MCMF graph. To the best of our knowledge, this article is the first attempt that aims at maximizing total reward for an EIP while limiting the power consumption of ESs by controlling their utilization.

III. System Architecture and Problem Formulation

A. System Architecture

Fig. 1 shows the MEC architecture that we are considering. ESs are allocated to one or more BSs to process tasks offloaded from IoT devices within these stations' coverage. Each IoT device may run several tasks, each of which can be offloaded to the ES separately. If a task is offloaded to an ES belonging to an EIP, then that EIP receives a reward which depends on the amount of computation performed by the server. Otherwise, offloaded tasks are forwarded to the central servers in the cloud.

The objective of this article is to maximize total reward while limiting power consumption over all ESs. To achieve this, we first determine the CPU utilization of each ES is determined to limit the total power for all ESs, then decide whether and where to offload each task to maximize the total reward as shown in Fig. 1.

B. Edge Server, Task, Power and Reward Model

1) Edge Server and Task Model: Many IoT applications, including surveillance, air quality monitoring, and traffic monitoring, require periodic collection of sensor data that needs to be processed periodically [8], [9]. For task offloading, each task calculates the amount of computation required over a period of time, and receives some proportion of the processor capacity provided by the ES. The task usage can be then expressed as the amount of computation required by the task divided by the total amount of computation provided by the ES

TABLE II
NOTATIONS USED IN THIS ARTICLE

Notations	Meaning
N^{ES}	Number of ESs
$C_j^{\max}$	Maximum processing capacity of ES j compared to the ES with the highest processing capacity
I^{high}	Index of the ES that has the highest processing capacity
U_i	Normalized usage of task τ_i
N^{task}	Number of total tasks
X_i	Index of the ES to which task τ_i is assigned
U_j^{total}	Total CPU utilization at ES j
Y_j	Maximum allowable CPU utilization for ES j
$P_j(u)$	Amount of power consumed at ES j when its CPU utilization is u, $(0 \leq u \leq 1)$
$H_{i,j}$	Binary constant to indicate whether task τ_i can be accepted by ES j
$R_{i,j}$	Reward received by the EIP for processing task τ_i at ES j
$R_{i,j}^{\mathrm{unit}}$	Reward per unit of processing capacity used at ES j
N_i^{sub}	Number of subtasks for task τ_i
$\tau_{i,m}$	mth subtask in task τ_i
$X_{i,m}$	Index of the ES to which subtask $\tau_{i,m}$ is assigned
$P_{i,j}^{\mathrm{algo}}$	Increase in the power drawn by ES j when a new task τ_i is allocated to it
$V_{i,j}^{\mathrm{algo}}$	$\frac{R_{i,j}}{P_{i,j}^{\mathrm{algo}}}$
$G=(V,E)$, G^{R}	Flow network and its residual network to represent edge allocation problem
$f(a,b)$, $C(a,b)$	Flow and cost between vertices a and b in a flow graph $G=(V,E)$
s, v_i, w_j, t	Source, task, ES and sink vertices in flow network G
$(\gamma^{\mathrm{T}},\delta^{\mathrm{T}})$	Range of task numbers per IoT device used for simulation
$(\gamma^{\mathrm{R}},\delta^{\mathrm{R}})$	Range of unit reward values used for simulation
$(\gamma^{\mathrm{U}},\delta^{\mathrm{U}})$	Ranges of U_i values used for simulation

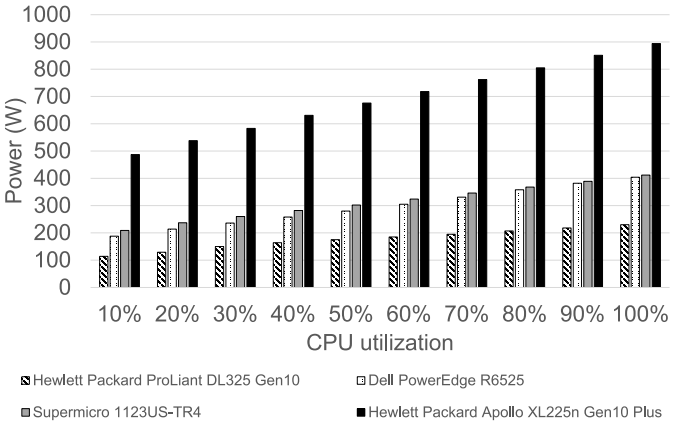

Fig. 2. Power consumption against CPU utilization for four different servers.

over a period of time. Then each task consumes its allocated processor capacity during each period. To represent this, it is assumed that time is divided by periods and that the ES allocation for each job is performed at the beginning of each period [40].

Since different ESs may have different processing capacities, we normalize usage of each task with respect to the ES with the highest processing capacity. If N^{ES} is the number of ESs, then we use $C_j^{\max}$, $(i=1,\ldots,N^{\mathrm{ES}})$ to denote the maximum processing capacity of ES j compared to the ES with the highest processing capacity, which has the index of I^{high}. Then, the normalized usage, U_i of task τ_i, $(i=1,\ldots,N^{\mathrm{task}})$ is determined on the basis that it is executed at ES I^{high}, where N^{task} is the total number of tasks. Assume that U_i and $C_j^{\max}$ have integer values. For this purpose, we introduce a grain of size to be the smallest unit of allocation and express the values of $C_j^{\max}$ and U_i as a multiple of this grain. For example, if this grain size is 0.01, then $C_{I^{\mathrm{high}}}^{\max}=100$. For ease of exposition, Table II summarizes notations used in this article.

2) Power Model: The CPU consumes the most power in the server, so most server power models use CPU utilization to derive overall system power [20], [22], [24]–[27]. Based on this, we assume that the server power consumption is dependent on CPU utilization. It has been shown that the relationship between power and CPU utilization is highly dependent on server characteristics [22], [23]. For example, the SPECpower committee [60] that produces tools and benchmarks for measuring the power consumption of servers have collected power measurements for servers and analyzed the relationship between power consumption and CPU utilization since 2006. For example, Fig. 2 gives data from the SPEC power collection for four different types of server. This data needs to be incorporated into any scheme for reducing the power consumption of ESs.

Let X_i, $(X_i=0,\ldots,N^{\mathrm{ES}})$ be the index of the ES to which task τ_i, $(i=1,\ldots,N^{\mathrm{task}})$ is assigned. We introduce an index 0 to represent central servers in the cloud, which process τ_i if it cannot be allocated to any ES. To summarize, if $X_i=0$, then τ_i is processed by the cloud; otherwise, if $X_i=j$, then τ_i is processed by ES j. The total usage, summed across all the tasks, cannot exceed the processing capacity so that $\sum_{\forall i,X_i=j} U_i \leq C_j^{\max}$. Thus, the total CPU utilization U_j^{total} at ES j, can be expressed as follows:

$$U_j^{\mathrm{total}}=\frac{\sum_{\forall i X_i=j} U_i}{C_j^{\max}}. \tag{1}$$

To limit the amount of power consumed by each ES, we limit its CPU utilization. For this purpose, we introduce the maximum allowable CPU utilization Y_j for each ES j, where Y_j is an integer value between 0 and $C_j^{\max}$. Then total CPU utilization at ES j must not exceed Y_j. If $Y_j=0$, then no task

can be processed by ES j. If $Y_j = C_j^{\max}$, then the processing capacity of ES j can be fully utilized.

The active power is the power consumed by a server while it is performing computations, and this varies nonlinearly with CPU utilization [60]; when no computation is being done, the server still draws a fixed idle power. We thus use $\alpha_j(u)$ to express the coefficient for active power when the CPU utilization is u, $(0 \leq u \leq 1)$ at ES j, and β_j to denote the coefficient for idle power for ES j. These values can easily be obtained [60]. Since the ratio of active to idle periods is u to $1-u$, we can define the function $P_j(u)$ to express the power drawn by ES j in terms of its utilization u as follows [60]:

$$P_j(u) = \alpha_j(u)u + \beta_j(1-u). \tag{2}$$

For example, for a Dell PowerEdge R6525 server, we found that: $\alpha_j(0.1) = 188$ W, $\alpha_j(0.2) = 214$ W, and $\beta_j = 81.6$ W [60]. Then, using (2), the power consumption is calculated as 92.24 W when the CPU utilization is 0.1, and 108.08 W when the CPU utilization is 0.2.

3) Reward Model: The EIPs can earn revenue by providing task offloading services for τ_i, but this also incurs costs from the use of computing resources on ESs. A reward then represents the net profit (i.e., revenue minus cost) if the task can be offloaded to the ES [14]. We use a linear model in which the values of revenue and cost for task τ_i are proportional to the amount of computation performed by the ES, U_i [14]. Thus, if a task is offloaded to an ES belonging to an EIP, then the EIP earns the net profit of $R_{i,j}$ proportional to U_i. We can then define the reward per unit of processing capacity used at ES j, as follows:

$$R_{i,j}^{\text{unit}} = \frac{R_{i,j}}{U_i}.$$

C. Problem Formulation

Our aim is now to allocate each task to an appropriate ES with the aim of maximizing overall reward under the following three constraints.

1) *ES Processing Capacity:* The total utilization at each ES j must not exceed its maximum allowable utilization

$$\forall j,\ U_j^{\text{total}} \leq Y_j. \tag{3}$$

2) *ES Coverage:* An ES can only accept tasks from devices within its coverage [16]. Therefore, we introduce the binary constant $H_{i,j}$ to indicate whether task τ_i can be accepted by ES j. If $H_{i,j} = 1$, then a request to offload task τ_i can be accepted by ES j; if $H_{i,j} = 0$, it cannot be accepted by ES j. Thus, a task τ_i can only be assigned to ES X_i if $H_{i,X_i} = 1$.
3) *Power:* P^{limit} is the maximum total power that all the ESs are permitted to consume

$$\sum_{j=1}^{N^{\text{ES}}} P_j\left(U_j^{\text{total}}\right) \leq P^{\text{limit}}. \tag{4}$$

We can then formulate the ES allocation and maximum allowable utilization determination (ESA-MAUD) problem, which consists of finding a value of X_i, $(X_i = 0, \ldots, N^{\text{ES}})$ for each task τ_i, and a maximum allowable utilization, Y_j, $(Y_j = 0, \ldots, C_j^{\max})$ for each ES j, with the aim of maximizing the total of all the servers' rewards

$$\begin{aligned}
\text{Maximize}\ & \sum_{i=1}^{N^{\text{task}}} R_{i,X_i} \\
\text{Subject to}\ & \forall i, H_{i,X_i} = 1,\ \left(i = 1, \ldots, N^{\text{task}}\right) \\
& \forall j, U_j^{\text{total}} \leq Y_j,\ \left(j = 1, \ldots, N^{\text{ES}}\right) \\
& \sum_{j=1}^{N^{\text{ES}}} P_j\left(U_j^{\text{total}}\right) \leq P^{\text{limit}}.
\end{aligned}$$

Theorem 1 states that this problem is NP-hard.

Theorem 1: The ESA-MAUD problem is NP-hard.

Proof: We prove the NP-hardness of this problem by a polynomial-time reduction from multiple knapsack problem with assignment restriction (MKAR), which is NP-hard [61], [62]. The MKAR allocates items to knapsacks for the purpose of maximizing total profit while satisfying knapsack capacity limits and assignment restrictions. In the ESA-MAUD problem, each task τ_i (item in MKAR) has a utilization, U_i (capacity in MKAR) and a reward, $R_{i,j}$, (profit in MKAR). Then it allocates each task to an appropriate ES (knapsack in MKAR) with the aim of maximizing total reward (total profit in MKAR) while satisfying ES processing capacity limit (knapsack capacity limit in MKAR) and ES coverage (assignment restriction in MKAR) constraints. Thus, the ESA-MAUD problem can be reduced to the MKAR, which is NP-hard. ■

IV. Maximum Allowable Utilization Determination

A. Algorithm Design

To solve the optimization problem, we need to determine the value of X_i for each task τ_i, $(i = 1, \ldots, N^{\text{task}})$, and the value of Y_j for each ES j, $(j = 1, \ldots, N^{\text{ES}})$. Each ES j has $C_j^{\max}$ utilizations, and it would be impractical to consider all the combinations of X_i and Y_j. We therefore divide this optimization problem into two subproblems, which we address successively. The utilization determination problem (UDP) is to find the values of maximum allowable utilization Y_j which meet the power constraint. The ES allocation problem (EAP) is to find the values of X_i which allocate tasks to ESs.

1) UDP: If the possible rewards at ES j, $\sum_{\forall i, X_i = j} R_{i,X_i}$ and corresponding power requirements are available, then UDP effectively reduces to a multiple-choice knapsack problem (MCKP) [63]. This requires an item, with a weight and a profit, to be selected from each of several classes containing multiple items, so as to maximize the total profit resulting from putting them in the knapsack without overloading it.

In the UDP, the classes are the ESs, each of which has $C_j^{\max}$ utilization values and each of them is associated with a power, which constitutes to a total P^{limit} that cannot be exceeded, and the reward to be maximized. MCKP is NP-hard, but a greedy algorithm based on profit to weight ratios

usually performs well [63]. This is the basis of our solution to UDP. Our algorithm repeatedly calculate the increases in power and reward from task allocations, giving priority to utilization values with higher reward to power ratios, until the power constraint in inequality (4) is satisfied, and the maximum allowable utilization Y_j of each ES j is determined.

2) EAP: We model the allocation of tasks to ESs using a MCMF graph [64], [65]. We consider two possible edge allocation algorithms: one that allows task splitting and one that does not. If task splitting is allowed, the subtasks created can be allocated to different ESs. In our model, a task τ_i may be split into N_i^{sub} subtasks, $\tau_{i,m}$, $(m = 1, \ldots, N_i^{\text{sub}})$. We constrain the CPU utilization of each subtask to have an integer value (like the utilization values of the tasks), and therefore the maximum number of subtasks into which task τ_i can be split is U_i. Let $X_{i,m}$, $(X_{i,m} = 0, \ldots, N^{\text{ES}})$ be the index of the ES to which subtask $\tau_{i,m}$ is allocated. An optimal set of integer-valued flows in an MCMF graph can be found [65], which minimizes the cost of delivering the maximum possible flow. We use this algorithm to determine the values of $X_{i,m}$, which correspond to these flows.

When task splitting is not allowed, so that each task must be executed on a single ES, EAP reduces to a multiple knapsack problem (MKP) [63], in which items with different weights and profit must be allocated to a number of knapsacks of limited capacity to maximize profit. In the EAP, the items are tasks and the aim is to maximize the total reward subject to the processing capacity of each ES. Like MKCP, MKP is NP-hard, and again we use a greedy algorithm, which allocates tasks to ESs on the basis of maximum reward, until no processing capacity remains.

B. Maximum Allowable Utilization Determination Algorithm

We must now determine the values of Y_j, for which we require three new variables. We first introduce a temporary variable X_i^{tmp} for each task τ_i to record the values of X_i during algorithm execution. Then $P_{i,j}^{\text{algo}}$ represents the increase in the power drawn by ES j when a new task τ_i is allocated to it, and this is expressed as follows:

$$P_{i,j}^{\text{algo}} = P_j\left(\frac{\sum_{\forall m X_m^{\text{tmp}}=j} U_m}{C_j^{\max}} + \frac{U_{i,j}}{C_j^{\max}}\right) - P_j\left(\frac{\sum_{\forall m X_m^{\text{tmp}}=j} U_m}{C_j^{\max}}\right) \tag{5}$$

where $P_j([\sum_{\forall m X_m^{\text{tmp}}=j} U_m]/[C_j^{\max}])$ is the power drawn by ES j before τ_i is allocated to ES j and $P_j([(\sum_{\forall m X_m^{\text{tmp}}=j} U_m)/(C_j^{\max})] + [U_{i,j}/C_j^{\max}])$ is the power drawn by ES j afterwards.

$V_{i,j}^{\text{algo}}$ is the specific reward, which represents the ratio between the increase in the reward resulting from the allocation of τ_i to ES j, and the increase in power formulated in (5). This specific reward is expressed as follows:

$$V_{i,j}^{\text{algo}} = \frac{R_{i,j}}{P_{i,j}^{\text{algo}}}. \tag{6}$$

The aim of the maximum allowable utilization determination (MUD) algorithm (Algorithm 1) is to obtain a large reward for a low power consumption. The MUD algorithm calculates the power requirement $P_{i,j}^{\text{algo}}$ and the specific reward $V_{i,j}^{\text{algo}}$ for each i and j. Next it determines the maximum $V_{i,j}^{\text{algo}}$, for which $i = M$ and $j = H$. It then marks τ_M as allocated by setting a temporary variable X_H^{tmp} to 1, and updates the total power requirement P^{used}. This procedure is repeated until the maximum power constraint can no longer be met, or no more tasks remain. As shown in Algorithm 1, the MUD algorithm repeatedly finds the maximum value of $V_{i,j}^{\text{algo}}$ until no more tasks remain in the worst case, which may require $([N^{\text{task}}(N^{\text{task}}+1)]/2)$ comparisons. Thus, its time complexity is $O(N^{\text{task}2})$.

Algorithm 1: MUD Algorithm

Output: Y_j, $(j = 1, ..., N^{\text{ES}})$;
Boolean variable: *flag* $\leftarrow$ *TRUE*;
Temporary variables: $P_{i,j}^{\text{algo}}$ and $V_{i,j}^{\text{algo}}$, $(i = 1, ..., N^{\text{task}})$, $(j = 1, ..., N^{\text{ES}})$;
Temporary variable: $P^{\text{used}} \leftarrow 0$;
Temporary variable: $\forall i, X_i^{\text{tmp}} \leftarrow 0$;
Set of task vertexes: $S^{\text{task}} = \{\tau_1, ..., \tau_{N^{\text{task}}}\}$;
while *flag* **do**
  for $\forall \tau_i \in S^{\text{task}}$ **do**
    Calculate $P_{i,j}^{\text{algo}}$ and $V_{i,j}^{\text{algo}}$ using Equations (5) and (6);
  end
  Find the highest value of $V_{i,j}^{\text{algo}}$, (say, $V_{H,M}^{\text{algo}}$);
  if $P^{\text{used}} + P_{H,M}^{\text{algo}} \leq P^{\text{limit}}$ **then**
    $P^{\text{used}} \leftarrow P^{\text{used}} + P_{H,M}^{\text{algo}}$;
    $X_H^{\text{tmp}} \leftarrow M$;
    $S^{\text{task}} \leftarrow S^{\text{task}} - \{\tau_H\}$;
  else
    flag $\leftarrow$ *FALSE*;
  end
  if $S^{\text{task}} = \phi$ **then**
    flag $\leftarrow$ *FALSE*;
  end
end
for $j = 1$ ***to*** N^{ES} **do**
  $Y_j \leftarrow \sum_{\forall i X_i^{\text{tmp}}=j} U_i$;
end

C. Algorithm Implementation

To obtain the values of Y_j for each ES j, the MUD algorithm runs on either one of the ESs, or a central cloud server. This master server needs three parameters [U_i, $R_{i,j}$, and $P_j(u)$] to derive the value of Y_j. The $P_j(u)$ values for most commercial servers are already available as provided by the SPECpower committee [60] through extensive measurements, the U_i values can be specified by each task τ_i, and $R_{i,j}$ is determined by the pricing policy.

After executing the MUD algorithm, the master server sends the value of Y_j to each ES j, limiting the CPU usage by ES j to Y_j. Many ES platforms employ these kinds of centralized

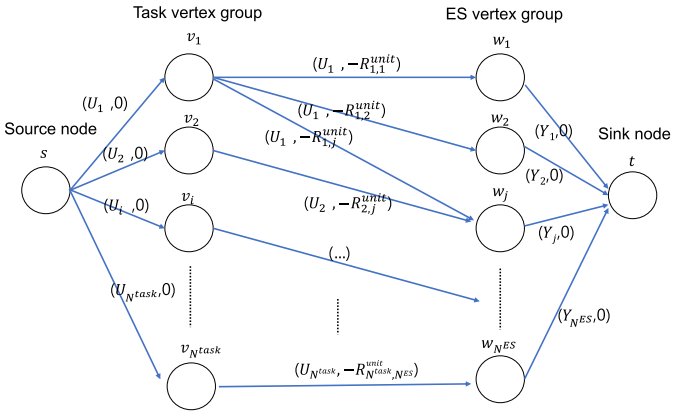

Fig. 3. Example of an MCMF graph representing an EAP.

management methods [66]. For example, on the Griffin ES platform [66], most decisions about ES management, such as resource provisioning are made in the central cloud.

After receiving the value of Y_j, each ES j limits its total CPU utilization by Y_j. Various methods can be used to limit the CPU utilization of each task [67]. For example, a Docker, an open source application container engine, provides a method for limiting the CPU time during each period so that the cpu-quota command can be used to set the CPU time allocated to each task within a period of cpu-period specified. Ubuntu Linux also provides a CPULimit Tool to limit the CPU usage of each task, or a group of tasks [68]. Based on these tools, the CPU utilization for each task can be effectively controlled, and its value can be obtained [67].

The MUD algorithm determines the values of Y_j to satisfy the power constraint only. Next, one of the ES allocation algorithms described in the next section can be executed to find the values of X_i for each task. After executing these algorithms on one of the ESs, or a central cloud server, the master server sends the value of Y_j to each ES j and X_i to each IoT device running task τ_i. The basic overhead of this approach is the algorithm execution, and there exist protocol overhead for delivering the values of Y_j and X_i and obtaining the initial values for [U_i, $R_{i,j}$ and $P_j(u)$] for algorithm execution.

Auction-based schemes can be used for many resource allocation problems three steps of bid submission, winner determination and reward determination [69]. To obtain the final reward value for each task to satisfy CPU computing capacity constraint ($U_j^{\text{total}} \leq Y_j$), they require Y_j values in advance for algorithm execution, but to obtain Y_j values, reward values are effectively used as can be seen in (6). These contradict each other.

V. Edge Server Allocation

A. MCMF Graph Construction

The EAP can be effectively represented as a MCMF problem [64], [65]. This is addressed by constructing an MCMF graph which is a flow network with source, sink and intermediate nodes, connected by directed edges. Source nodes have only outgoing edges, and sink nodes only have incoming edges. An edge from node a to node b has a capacity $S(a, b)$ and cost $C(a, b)$. The MCMF problem is then to find a set of flows from source to sink nodes, which maximizes the total flow at the lowest cost.

EAP can be modeled as a flow network with four types of vertex as shown in Fig. 3: a single source vertex s, a task vertexes $v_1, \ldots, v_{N^{\text{task}}}$, where v_i is corresponds to task τ_i, an ES vertexes $w_1, \ldots, w_{N^{\text{ES}}}$, where w_j corresponds to ES j, and a single sink vertex t. Source and sink are just meta vertices expressing EAP by MCMF, so they are not physically mapped to any component of EAP. The rules used to construct flow network can be described as follows.

1) The source vertex s is connected to all the task vertexes $v_1, \ldots, v_{N^{\text{task}}}$. Each edge (s, v_i) has a capacity U_i and a cost of 0.
2) Each vertex v_i, $(i = 1, \ldots, N^{\text{task}})$ is connected to the vertices w_j for which $H_{i,j} = 1$. Each edge (v_i, w_j) has a capacity U_i and a cost, $-R_{i,j}^{\text{unit}}$.
3) Each vertex w_j, $(j = 1, \ldots, N^{\text{ES}})$ is connected to the sink vertex t. Each edge (w_j, t) has a capacity Y_j and a cost of 0.

The EAP is solved by determining the values of $X_{i,m}$, when task splitting is allowed, and X_i, when task splitting is not allowed, with the aim of maximizing the total reward while meeting the utilization constraint at each ES. Maximizing the total reward corresponds to minimizing the total cost $\sum -R_{i,Y_j}^{\text{unit}} U_i$. If the flow capacity for task τ_i from source to sink node is U_i, then the total flow from each ES vertex w_j to the sink vertex cannot exceed the capacity constraint of ES j, which is Y_j. The ES coverage constraint is represented in the MCMF graph because there edges between v_i and w_j are only present if $H_{i,j} = 1$.

B. Edge Server Allocation When Task Splitting Is Allowed

The ES allocation algorithm when task splitting is allowed (EAA-TS) is shown as Algorithm 2. EAA-TS searches for paths from source to sink with decreasing cost which also meet the capacity constraint, using the shortest path faster algorithm (SPFA), which is an improvement on the well-known Bellman-Ford algorithm [70].

We now introduce the binary function $f(a, b)$ to represent the flow between vertices a and b: if the value of $f(a, b)$ is 1, then there is a flow. The path chosen during algorithm execution is not part of the final flow, so the SPFA needs a reverse path, which can be undone [65], [70]. Therefore, we must construct a residual graph, in which each edge represents additional flow that can be allowed between two vertices and the reverse paths are allowed.

For example, suppose that $f(v_1, w_1) = 1$ in Fig. 3, so that there is a flow from v_1 to w_1. The corresponding residual graph is shown as Fig. 4. The capacity values for all the edges $s \to v_1 \to w_1 \to t$ are decremented by 1, but the capacities of edges on the reverse path are incremented by 1. The cost $-R_{v_1,w_1}^{\text{unit}}$ for this reverse path, $C(w_1, v_1)$ is set to $-C(v_1, w_1)$, which is $R_{v_1,w_1}^{\text{unit}}$. The reverse path may be selected by the algorithm, and then the original path selection is canceled. SPFA operates on the residual graph until there remains no feasible flow.

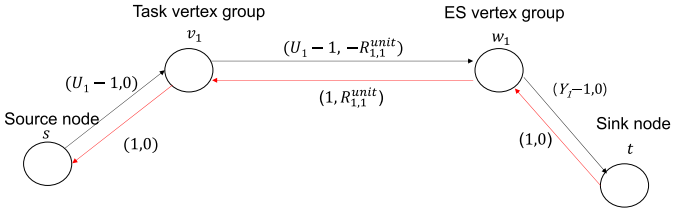

Fig. 4. Example of residual graph. The red lines show the reverse path.

EAA-TS includes initialization, path augmentation and finalization steps which run before and after SPFA as shown in Algorithm 2.

1) *Initialization:* A residual graph G^R is constructed from the MCMF graph (line 1), and then the values of flow $f(a, b)$ and cost $C(a, b)$ are initialized (lines 5–9).
2) *Path Augmentation:* A minimum-cost flow is found using SPFA (line 11); its path is augmented in the residual graph (line 15); and then the flows are updated (lines 17 and 18). This step is repeated until there is no augmenting path p in G^R (line 12).
3) *Finalization:* If there is a flow between v_i and w_j, then ES j is chosen to process one of the subtasks in τ_i, and the values of $X_{i,m}$ are updated accordingly (lines 21–29).

To derive time complexity of Algorithm 2, consider a flow network, $G = (V, E)$ in Fig. 3, where V and E are sets representing the vertices and edges of the flow network, respectively. SPFA is known to have a $O(|V||E|)$ time complexity in the worst case and $O(|E|)$ in the average case [70]. In addition, since all the flows between the source and sink vertices may be examined, the time complexity in the worst case can be calculated as: $O(|V||E| \sum_{i=1}^{N^{\mathrm{ES}}} Y_j)$ [64], [65].

C. Edge Allocation Algorithm Without Task Splitting

The edge allocation algorithm with no task splitting (EAA-NTS) (Algorithm 3) operates on the MCMF graph as follows.

1) *Minimum Cost Flow:* The minimum-cost flow f from source to sink vertices is found (line 5).
2) *Update and Elimination:* The task vertex associated with the minimum-cost flow is eliminated, the capacity usage C_j^{tmp} and the index X_i are updated, and the edges associated with the flow are eliminated (lines 6–16). For example, suppose that the minimum-cost flow is $s \rightarrow v_1 \rightarrow w_2 \rightarrow t$ in Fig. 3. Then the vertex corresponding to τ_1 is eliminated, together with its incoming and outgoing edges. The value of X_1 is updated to 2 because τ_1 is allocated to ES 2. The temporary variable C_2^{tmp} is increased by U_1 to record the increased processing usage by ES 2, which must not exceed Y_j. These two steps are repeated until all the tasks have been examined.

EAA-NTS repeatedly finds the minimum-cost flow one by one until no task left in S^{task}, which involves sorting tasks in nondecreasing order of $-R_{i,j}^{\mathrm{unit}}$. This, if quick sorting is used, the time complexity for Algorithm 3 is $O(N^{\mathrm{task}} \log N^{\mathrm{task}})$ on average [64].

Algorithm 2: Edge Allocation Algorithm With Task Splitting (EAA-TS)

Input: A flow network G in Fig. 3;
Output: $\forall i, m$, $X_{i,m}$ for $\tau_{i,m}$;
1 Residual network G^{R} of G;
2 Temporary variables for all the edges, (a, b) in network G^{R}: $f(a, b)$, $f(b, a)$, $C(a, b)$ and $C(b, a)$;
3 Temporary variable: $I_i^{\mathrm{sub}} \leftarrow 1$, $(i = 1, ..., N^{\mathrm{task}})$;
4 Construct a residual graph G^{R} based on G;
5 **for** *all edges* (a, b) *in* G^R **do**
6 $\quad C(b, a) \leftarrow -C(u, z)$;
7 $\quad f(a, b) \leftarrow 0$;
8 $\quad f(b, a) \leftarrow 0$;
9 **end**
10 **while** *TRUE* **do**
11 $\quad$ Run the SPFA algorithm to find an augmenting path p from s to t using the minimum cost from G^{R};
12 $\quad$ **if** *there is no augmenting path p from* G^{R} **then**
13 $\quad\quad$ break;
14 $\quad$ **end**
15 $\quad$ Augment all the flows along p to network G^{R};
16 $\quad$ **for** *all edges* $(a, b) \in p$ **do**
17 $\quad\quad$ Update the flow, $f(a, b)$ based on flow augmentation;
18 $\quad\quad$ Update the flow, $f(b, a)$ based on flow augmentation;
19 $\quad$ **end**
20 **end**
21 **for** $j = 1$ to $j = N^{\mathrm{es}}$ **do**
22 $\quad$ **for** *all edges from task to ES vertices,* (v_i, w_j) *in* G^R **do**
23 $\quad\quad$ **while** $f(v_i, w_j) > 0$ **do**
24 $\quad\quad\quad$ $X_{i,I_i^{\mathrm{sub}}} \leftarrow j$;
25 $\quad\quad\quad$ $I_i^{\mathrm{sub}} \leftarrow I_i^{\mathrm{sub}} + 1$;
26 $\quad\quad\quad$ $f(v_i, w_j) \leftarrow f(v_i, w_j) - 1$;
27 $\quad\quad$ **end**
28 $\quad$ **end**
29 **end**

VI. Experimental Results

A. Simulation Setup

We performed simulations to evaluate our scheme in terms of power consumption and total reward. We modeled the power characteristics of four commercial servers released between 2019 and 2020 [60] as shown in Fig. 2. The characteristics of one of these commercial servers is allocated to each ES at random.

We modeled the spatial relationship between ESs and IoT devices using a database of cellular network location data from Melbourne Australia [15], [16]. This contains the latitudes, longitudes and IP addresses of 815 WAPs and 125 BSs, from which distances between WAPs and each BS can be calculated.

Unless otherwise stated, we will assume that each BS has a single ES to handle tasks within its coverage. And we will take the locations of WAPs in the Melbourne database to represent

TABLE III
PARAMETER SETTINGS

Parameters	Range description	Default range	Ranges used in the experiments
Number of tasks per IoT device	(γ^{T}, δ^{T})	(5,10)	(3,6), (5,10), (7,14), (9,18)
Reward	(γ^{R}, δ^{R})	(1,15)	(1,5), (1,15), (1,25), (1,35)
Task usage, U_i	(γ^{U}, δ^{U})	(3,8)	(1,6), (3,8), (5,10), (7,12), (1,12)

Algorithm 3: Edge Allocation Algorithm for No Task Splitting (EAA-NTS)

Input: A flow network G in Fig. 3;
Output: X_i for τ_i, $(i = 1, ..., N^{\mathrm{task}})$;
Temporary variable: $C_j^{\mathrm{tmp}} \leftarrow 0$, $(j = 1, ..., N^{\mathrm{ES}})$;
$S^{\mathrm{task}} \leftarrow \{v_1, ..., v_{N^{\mathrm{task}}}\}$;
$\forall i, X_i \leftarrow 0$;
while $S^{\mathrm{task}} \neq \phi$ **do**
  Find the minimum-cost flow with the lowest values of $-R_{i,j}^{\mathrm{unit}}$, (say, $s \rightarrow v_L \rightarrow w_M \rightarrow t$);
  if $C_M^{\mathrm{tmp}} + U_L \leq Y_j$ **then**
    $X_M \leftarrow M$;
    $C_M^{\mathrm{tmp}} \leftarrow C_M^{\mathrm{tmp}} + U_L$;
    Eliminate an edge from s to v_L and all the outgoing edges from v_L;
    $S^{\mathrm{task}} \leftarrow S^{\mathrm{task}} - \{v_L\}$;
  else
    Eliminate an outgoing edge from v_L to w_M;
    if *there is no outgoing edge from* v_L **then**
      $S^{\mathrm{task}} \leftarrow S^{\mathrm{task}} - \{v_L\}$;
    end
  end
end

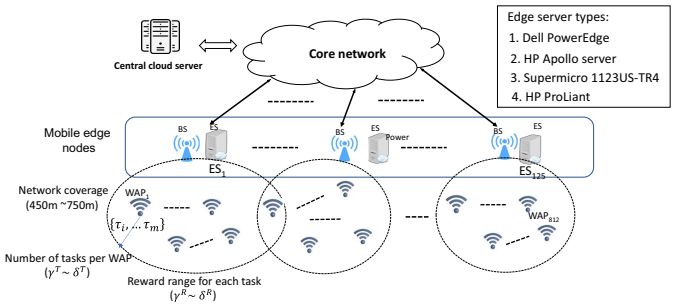

Fig. 5. Simulation scenario based on real celluar network model [15], [16].

the locations of IoT devices. If the WAP connected to a device that wishes to offload task τ_i is within the coverage of the BS connected to ES j, then $H_{i,j} = 1$ (otherwise 0). The coverage of each BS is represented by distances randomly chosen between 450 m and 750 m as described in [15] and [16].

To examine the effects of various parameters, we varied number of tasks, rewards and task usage, each of which has different ranges as tabulated in Table III. Each WAP can be connected to IoT devices generating a random number of tasks between γ^T and δ^T; the unit reward value for each task τ_i, $R_{i,j}^{\mathrm{unit}}$ which is the same for every ES j, is a randomly chosen integer between γ^R and δ^R; and the usage U_i required by task τ_i is a randomly chosen integer between γ^U and δ^U. Except when otherwise stated, the default range for each parameter in Table III is used.

We assess two schemes: 1) EAA-TS (Algorithm 2) and 2) EAA-NTS (Algorithm 3), each of which used the MUD algorithm (Algorithm 1) to determine the maximum allowable utilization of each ES. We compare these schemes with the following four methods of assigning tasks to ESs while limiting power consumption to satisfy inequality (4).

1) RR+HR (Round-robin allocation + highest reward) allocates tasks in a round-robin fashion, successively choosing the tasks that produces the highest reward until inequality (4) is no longer satisfied.
2) RA+HR (random task allocation + highest reward) chooses the allocation with the highest reward for successive randomly selected tasks until inequality (4) is no longer satisfied.
3) MUD+HR (MUD + highest reward) is similar to a MUD algorithm, except that the value of $V_{i,j}^{\mathrm{algo}}$ is set to $R_{i,j}$ so that the task allocation producing the highest reward can be performed first. Values of X_i and Y_j are successively updated until inequality (4) is no longer satisfied.
4) MUD+HUR (MUD + highest unit reward) is similar to MUD+HR, except that the value of $V_{i,j}^{\mathrm{algo}}$ is set to $R_{i,j}^{\mathrm{unit}}$. Values of X_i and Y_j are successively updated until inequality (4) is no longer satisfied.
5) LPF+HR (lowest-power-first + highest reward) All Y_j values are initialized to 0. Then it finds the ES with the lowest power as a result of increasing the utilization rate by 1 and increments its Y_j value by 1. This step is repeated until inequality (4) is no longer satisfied. After determining Y_j values, it makes use of the main idea of the successive knapsack algorithm [61], [62], so selects task with the highest reward successively for each ES while inequality (3) is being satisfied.

The first two baseline schemes attempt to increase total reward while balancing workloads; variants of the MUD algorithm, (MUD+HR and MUD+HUR) aim to maximize the highest reward and the highest ratio of reward to usage, respectively; LPF+HR aims for achieving the highest reward while taking power consumption into account.

B. Effect of Task Splitting

We express power limit values as percentage of the total power consumption when all the ESs are fully utilized, which is expressed by P^{ratio}. To examine the effect of task splitting on total reward, we compared EAA-TS with EAA-NTS for all combinations of (γ^T, δ^T), (γ^R, δ^R), and (γ^U, δ^U) as shown in Table III when P^{ratio} are 40%, 60%, and 80% (total 240 results for each scheme). Table IV shows the average and maximum percentage differences of total reward between the results for

TABLE IV
COMPARISON BETWEEN EAA-NTS AND EAA-TS FOR DIFFERENT VALUES OF (γ^U, δ^U) AND P^{RATIO}

Average percentage difference of total reward.					
P^{ratio}	(γ^U, δ^U)				
	(1,6)	(1,12)	(3,8)	(5,10)	(7,12)
40%	0.01%	0.07%	2.92%	6.95%	7.01%
60%	0.06%	0.04%	1.84%	3.63%	5.60%
80%	0.02%	0.01%	1.15%	2.92%	4.31%
Maximum percentage difference of total reward.					
P^{ratio}	(γ^U, δ^U)				
	(1,6)	(1,12)	(3,8)	(5,10)	(7,12)
40%	0.04%	0.25%	3.54%	8.66%	9.07%
60%	0.48%	0.33%	2.12%	4.04%	6.18%
80%	0.07%	0.07%	1.15%	3.2%	5.01%

EAA-TS and EAA-NTS for different values of (γ^U, δ^U) and P^{ratio}.

We found that when $\gamma^U = 1$, the percentage difference between the results for EAA-TS and EAA-NTS is negligible regardless of the value of the other parameters; for example, it never exceeds 0.48% when $\gamma^U = 1$. However, as γ^U increases, a gap opens up between them. This can be explained as follows: EAA-NTS requires a task to be allocated to a single ES, so a task cannot be allocated to ES with insufficient remaining capacity to process that task as a whole, resulting in low utilization, which occurs when the minimum task usage size exceeds 1. As P^{ratio} increases, the gap between them decreases, suggesting that task splitting is effective when available power is low.

C. Effect of Power Limit on Total Reward

We examined the effect of the power limit on the total reward. Fig. 6 shows rewards for all schemes normalized to those for EAA-TS. Higher power limits produce greater rewards. In addition, EAA-TS yields the highest reward for all power limits; these rewards are 8%–80% higher than those for the other schemes. This disparity is greater at lower power limits. The MUD+HUR scheme produces higher rewards than MUD+HR, suggesting that tasks with higher value of $R^{\text{unit}}_{i,j}$ (rather than $R_{i,j}$) need to be allocated first. However, MUD+HUR takes no account of edge sever characteristics: it only produces an average of 68% of the reward of EAA-TS, and 39% at the lowest power limit. LPF+HR produces 81% of the reward of EAA-TS, yielding a higher reward than any other alternatives, but shows worse performance than MUD+HUR when $P^{\text{ratio}} = 80\%$.

D. Total Reward Against the Number of Tasks

We examined how total reward depends on the number of tasks generated within each IoT device. Fig. 7 shows the average reward results for three P^{ratio} values: 40%, 60%, and 80%. Again EAA-TS and EAA-NTS outperform the other schemes, and the difference tends to increase as the number of tasks increases. We attribute this to the availability of more tasks with high rewards for early selection. For example, whereas the reward obtained by EAA-TS increases by 5119 as the range of task numbers varies from (3, 6) to (9, 18), the reward obtained by MUD+HUR increases by 2684.

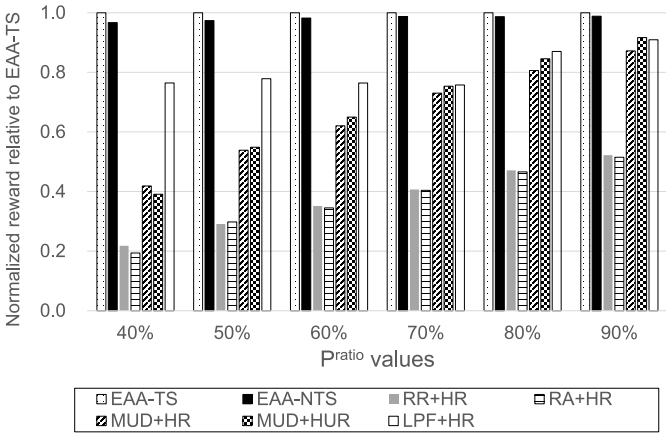

Fig. 6. Total reward, normalized against results from EAA-TS, against normalized power limit.

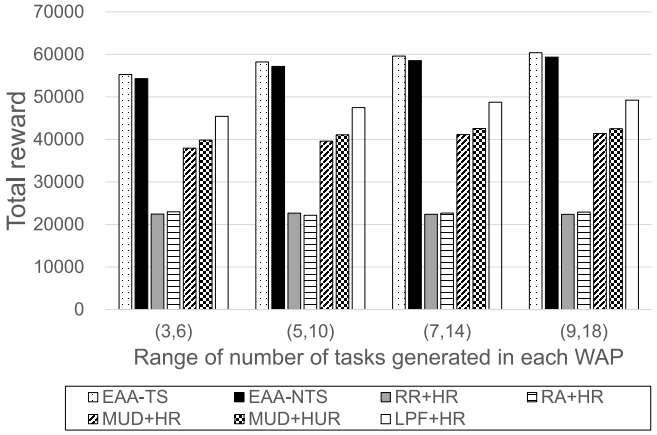

Fig. 7. Average reward against values of (γ^T, δ^T).

E. Effect of Reward Ranges

We looked at the effect of different reward ranges, (γ^R, δ^R). Fig. 8 shows the average reward results when P^{ratio} are 40%, 60%, and 80%. Again EAA-TS and EAA-NTS outperforms all the other scheme with a growing margin as the gap between γ^R and δ^R increases. Having a wide range of rewards attached to tasks gives more scope for optimization, and it appears that our schemes are able to take advantage of this.

F. Effect of Task Usage Range

We also examined how the total reward varies over a range of task usage values. Fig. 9 shows total reward for a number of different ranges, (γ^U, δ^U), averaged over three P^{ratio} values: 40%, 60%, and 80%. Except the basic schemes, such as RR+HR and RA+HR, total reward increases slightly as the range of (γ^U, δ^U) increases, because fewer tasks can be allocated to ESs, but these tasks carry higher rewards, but this increase gradually tails off. The difference between the results for EAA-TS and EAA-NTS increases as the range of (γ^U, δ^U) increases.

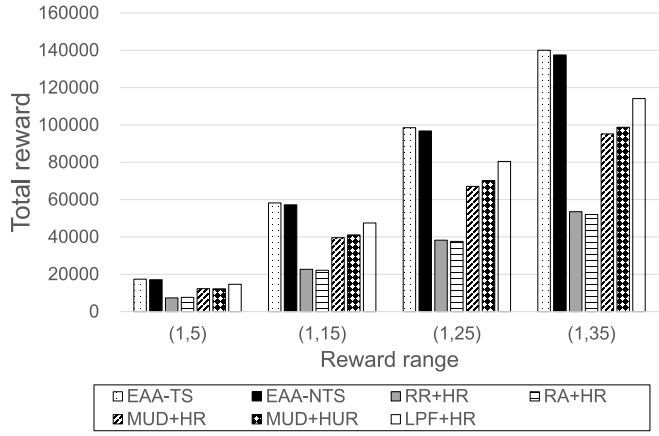

Fig. 8. Average reward against values of (γ^R, δ^R).

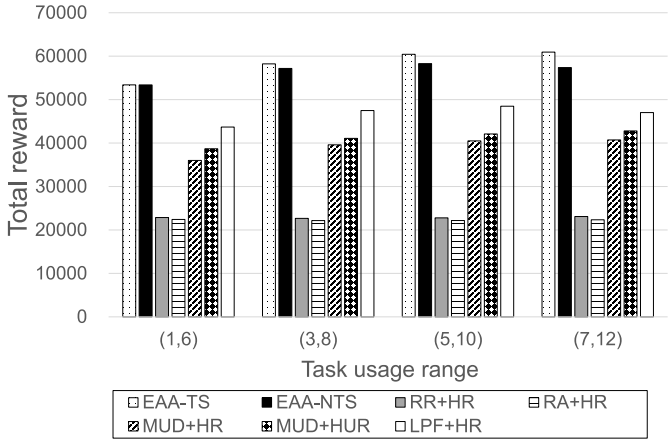

Fig. 9. Average reward against values of (γ^U, δ^U).

G. Power Consumption Required by Different Schemes to Achieve the Same Reward as EAA-TS

We determined the total reward obtained by EAA-TS for different values of P^{ratio}. We then found the power required by each of the other schemes to achieve the same reward. Fig. 10 shows the results, normalized to the power drawn by EAA-TS in each case. To achieve the same rewards, the other schemes consume more power that EAA-TS and EAA-NTS. For example, MUD+HUR draws between 12% and 29% more power than EAA-TS and LPF+HR draws 8% and 25% more. When $P^{\text{ratio}} = 60\%$, 70%, and 80%, RR+HR and RA+HR fail to produce the same reward as EAA-NTS, even though all the ESs are fully utilized.

H. Effect of Number of Edge Servers

We also examined how the total reward varies against number of ESs by increasing the number of ESs connected to each BS. Fig. 11 shows how total reward depends on the values of N^{ES}, averaged over three P^{ratio} values: 40%, 60%, and 80%. The total reward increases with the value of N^{ES}, because more tasks can be offloaded to the ESs. However, the value of N^{ES} hardly affects the percentage difference between EAA-TS and the other schemes. For example, the percentage difference between LPR+HR and EAA-TS is between 18% and 20% for all values of N^{ES}.

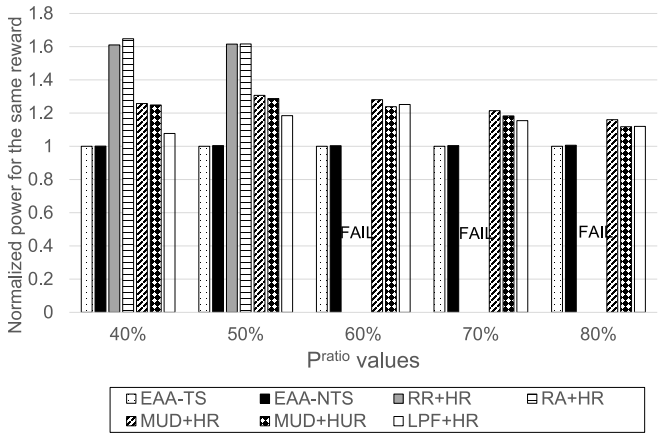

Fig. 10. Power consumption, normalized against the power required by EAA-NTS to achieve the same total reward as EAA-NTS, for different values of P^{ratio}.

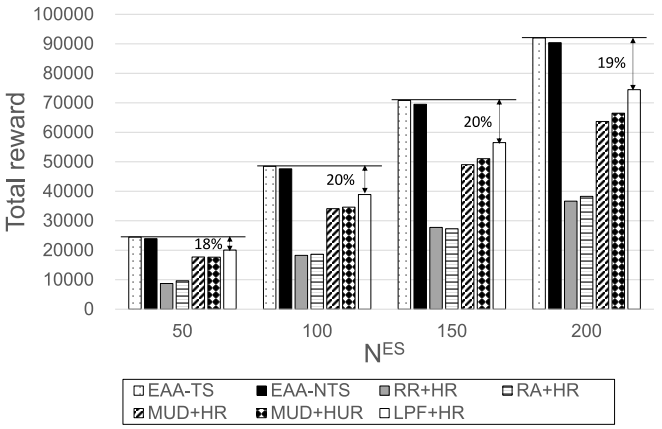

Fig. 11. Average reward against values of N^{ES}.

TABLE V
AVERAGE EXECUTE TIME (S) OF THE MUD, EAA-TS, AND EAA-NTS ALGORITHMS AGAINST VALUES OF (γ^T, δ^T)

$(\gamma^{\text{T}}, \delta^{\text{T}})$	(3,6)	(5,10)	(7,14)	(9,18)
N^{task}	3,263	5,720	8,201	10,537
MUD	0.241	0.564	0.921	1.394
EAA-NTS	0.101	0.105	0.109	0.114
EAA-TS	0.493	1.189	1.826	2.905

I. Algorithm Execution Time Analysis

We measured the algorithm execution times 5 times on the Intel i7-10700 CPU and Table V shows the average results for different values of (γ^T, δ^T), averaged over three P^{ratio} values of 40%, 60%, and 80%. Each algorithm can run within a reasonable amount of time, and the execution time increases with the values of N^{task} because more computation is required to support more tasks, but this effect on EAA-NTS seems to be almost negligible. In addition, EAA-TS allows task splitting to find an optimal solution, but takes longer than EAA-NTS.

J. Discussion

We draw the following conclusions from our evaluation.

1) Our schemes achieve between 8% and 80% higher rewards than the other schemes, under the same power constraints. In particular, EAA-TS produces 10%–24% more rewards than LPF-HR that achieves the highest reward among the other schemes. In the real world, these results indicate that our schemes return more economical profit to EIP.
2) To reduce the load on power distribution infrastructure, power capping is essential, which regulates the amount of power consumed by ESs during periods of peak demand [21]. The results showed that our schemes are especially effective at lower power limits, suggesting that they can effectively cope with the increasing demand for offloading workloads under power limit conditions.
3) Among the various parameters, only the value of task usage size for each task affects the resulting difference between EAA-TS and EAA-NTS. For example, their differences are almost negligible when $\gamma^U = 1$, but increases as γ^U increases. These suggest that task splitting can be effective when there are many tasks with high CPU utilization.
4) Increasing N^{task} allows more tasks with higher reward to be selected thereby increasing total reward, but this effect is small.
5) Our schemes effectively handle situations where the rewards assigned to each task are variable, giving the EIP more revenue than other alternatives.

VII. Conclusion

We have proposed a scheme to maximize the total reward accrued by ESs, while limiting the power that they consume. We first examined how the processor capacity of an each ES and the power that it draws, depend on its utilization. We then formulated an optimization problem that determines the maximum allowable utilization of each ES and a task allocation with the aim of maximizing the total reward. To determine the maximum allowable utilization, we used a greedy algorithm that tries to achieve the highest reward and the lowest total power consumption by the ESs. We formulated the task allocation problem as a MCMF graph and proposed two algorithms to find the highest reward paths in the graph. One algorithm allows task splitting and the other does not.

We conducted simulations to assess our scheme in terms of total reward, while varying the number of tasks, power limit values, reward and task usage ranges, and the ratio of reward to power consumption. Experimental results showed that our scheme makes effective use of available power to obtain a large reward. Its rewards were between 8% and 80% larger than those obtained by alternative schemes, and these differences require other schemes to consume 8% and 65% more power than ours to achieve the same rewards. Our scheme is particularly effective when reduced power is available, when the rewards assigned to each task are very variable, and when there are many tasks. Task splitting makes little difference when the minimum task usage size is 1 but is effective when an average task requires a high proportion of a CPU's processing capacity.

The demand for edge computing is increasing rapidly to support growing numbers of IoT tasks, resulting in high power consumption by ESs. The results we have presented show that our scheme can contribute to the effective power management of ESs. In future work, we plan to extend our scheme to take various QoS factors into account. It would be interesting to apply auction-based techniques to obtain maximum reward subject to power limitation using a compromise process.

References

[1] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 450–465, Feb. 2018.

[2] L. U. Khan, I. Yaqoob, N. H. Tran, S. M. A. Kazmi, T. N. Dang, and C. S. Hong, "Edge-computing-enabled smart cities: A comprehensive survey," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 10200–10232, Oct. 2020.

[3] C. Jiang, X. Cheng, H. Gao, X. Zhou, and J. Wan, "Toward computation offloading in edge computing: A survey," *IEEE Access*, vol. 7, pp. 131543–131558, 2019.

[4] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu, and W. Zhuang, "Learning-based computation offloading for IoT devices with energy harvesting," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1930–1941, Feb. 2019.

[5] J. Wang, J. Hu, G. Min, W. Zhan, Q. Ni, and N. Georgalas, "Computation offloading in multi-access edge computing using a deep sequential model based on reinforcement learning," *IEEE Commun. Mag.*, vol. 57, no. 5, pp. 64–69, May 2019.

[6] A. Samanta, Z. Chang, and Z. Han, "Latency-oblivious distributed task scheduling for mobile edge computing," in *Proc. IEEE Global Commun. Conf.*, Dec. 2018, pp. 1–7.

[7] T. Zhao, S. Zhou, X. Guo, and Z. Niu, "Tasks scheduling and resource allocation in heterogeneous cloud for delay-bounded mobile edge computing," in *Proc. ICC*, May 2017, pp. 1–7.

[8] S.Jošilo and G. Dan, "Computation offloading scheduling for periodic tasks in mobile edge computing," *IEEE/ACM Trans. Netw.*, vol. 28, no. 2, pp. 667–680, Apr. 2020.

[9] N. Min-Allah, M. B. Qureshi, F. Jan, S. Alrashed, and J. Taheri, "Deployment of real-time systems in the cloud environment," *J. Supercomput.*, vol. 77, no. 4, pp. 2069–2090, 2021.

[10] *12 Gartner Edge Computing Use Cases We Believe Can Help You Win at Edge Computing*. Accessed: Mar. 16, 2021. [Online]. Available: https://blog.stratus.com/12-gartner-edge-computing-use-cases-win-edge-computing/

[11] S. Wang, Y. Zhao, J. Yuand, and C.-H. Hsu, "Edge server placement in mobile edge computing," *J. Parallel Distrib. Comput.*, vol. 127, pp. 160–168, May 2019.

[12] T. Lahderanta, T. Leppanen, L. Ruha, L. Loven, E. Harjula, and M. Ylianttila, "Edge server placement with capacitated location allocation," Mar. 2020. [Online]. Available: arXiv:1907.07349.

[13] C. Anglano, M. Canonico, and M. Guazzone, "Profit-aware resource management for edge computing system," in *Proc. ACM Int. Workshop Edge Syst. Anal. Netw.*, May 2018, pp. 25–30.

[14] D. Nguyen, L. B. Le, and V. Bhargava, "Price-based resource allocation for edge computing: A market equilibrium approach," *IEEE Trans. Cloud Comput.*, vol. 9, no. 1, pp. 302–317, Jan.–Mar. 2021.

[15] P. Lai *et al.*, "Optimal edge user allocation in edge computing with variable sized vector bin packing," in *Proc. IEEE Int. Conf. Service-Oriented Comput.*, Nov. 2018, pp. 230–245.

[16] Q. He *et al.*, "A game-theoretical approach for user allocation in edge computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 3, pp. 515–529, Mar. 2020.

[17] Y. Hao, L. Hu, Y. Qian, and M. Chen, "Profit maximization for video caching and processing in edge cloud," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 7, pp. 1632–1641, Jul. 2019.

[18] E. Ahvar, A.-C. Orgerie, and A. Lebre, "Estimating energy consumption of cloud, fog and edge computing infrastructures," *IEEE Trans. Sustain. Comput.*, early access, Mar. 18, 2019, doi: 10.1109/TSUSC.2019.2905900.

[19] Y. Li and S. Wang, "An energy-aware edge server placement algorithm in mobile edge computing," in *Proc. IEEE Int. Conf. Edge Comput.*, Jul. 2018, pp. 66–73.

[20] M. Daraghmeh, I. A. Ridhawi, M. Aloqaily, Y. Jararweh, and A. Agarwal, "A power management approach to reduce energy consumption for edge computing servers," in *Proc. IEEE Int. Conf. Fog Mobile Edge Comput.*, Jun. 2019, pp. 259–264.

[21] Y. Li *et al.*, "A scalable priority-aware approach to managing data center server power," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, Feb. 2019, pp. 149–164.

[22] W. Lin, W. Yu, H. Wang, J. Z. Wang, and C.-H. Hsu, "Experimental and quantitative analysis of server power model for cloud data centers," *Future Gener. Comput. Syst.*, vol. 86, pp. 940–950, Sep. 2018.

[23] Y. Wang, D. Nortershauser, S. Masson, and J. Menaud, "An empirical studay of power characterization approaches for servers," in *Proc. Int. Conf. Smart Grids Green Commun. IT Energy Aware Technol.*, Jun. 2019, pp. 1–6.

[24] M. Dayarathna, Y. Wen, and R. Fen, "Data center energy consumption modeling: A survey," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 732–794, 1st Quart., 2016.

[25] X. Zhang, J.-J. Lu, X. Qin, and X.-N. Zhao, "A high-level energy consumption model for heterogeneous data centers," *Simulat. Model. Pract. Theory*, vol. 39, pp. 41–55, Dec. 2013.

[26] M. Tang and S. Pan, "A hybrid genetic algorithm for the energy-efficient virtual machine placement problem in data centers," *Neural Process. Lett.*, vol. 41, no. 2, pp. 211–221, 2015.

[27] V. Gupta, R. Nathuji, and K. Schwan, "An analysis of power reduction in datacenters using heterogeneous chip multiprocessors," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 39, no. 3, pp. 87–91, 2011.

[28] M. Song, Y. Lee, and J. Park, "Scheduling a video transcoding server to save energy," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 11, no. 2s, p. 45, Feb. 2015.

[29] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Trans. Commun.*, vol. 64, no. 10, pp. 4268–4283, Oct. 2016.

[30] Y. Wang, M. Sheng, X. Wang, and J. Li, "Cooperative dynamic voltage scaling and radio resource allocation for energy-efficient multiuser mobile edge computing," in *Proc. IEEE Int. Conf. Commun.*, Jul. 2018, pp. 1–6.

[31] I. Sarrigiannis, K. Ramantas, E. Kartsakli, P. Mekikis, A. Antonopoulos, and C. Verikoukis, "Online VNF lifecycle management in an MEC-enabled 5G IoT architecture," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4183–4194, May 2020.

[32] I. Sarrigiannis, L. Contreras, K. Ramantas, A. Antonopoulos, and C. Verikoukis, "Fog-enabled scalable C-V2X architecture for distributed 5G and beyond applications," *IEEE Netw.*, vol. 34, no. 5, pp. 120–126, Sep./Oct. 2020.

[33] J. Sun, Q. Gu, T. Zheng, P. Dong, A. Valera, and Y. Qin, "Joint optimization of computation offloading and task scheduling in vehicular edge computing networks," *IEEE Access*, vol. 8, pp. 10466–10477, 2020.

[34] J. Ren, G. Yu, Y. He, and G. Y. Li, "Collaborative cloud and edge computing for latency minimization," *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 5031–5044, May 2019.

[35] X. Xu *et al.*, "Load-aware edge server placement for mobile edge computing in 5G networks," in *Proc. IEEE Int. Conf. Service-Oriented Comput.*, Oct. 2019, pp. 494–507.

[36] J. Xu, B. Palanisamy, H. Ludwig, and Q. Wang, "Zenith: Utility-aware resource allocation for edge computing," in *Proc. IEEE Int. Conf. Edge Comput.*, Jun. 2017, pp. 47–54.

[37] N. Riaz, S. Qaisar, M. Ali, and M. Naeem, "Node selection and utility maximization for mobile edge computing-driven IoT," *Trans. Emerg. Telecommun. Technol.*, to be published, [Online]. Available: https://doi.org/10.1002/ett.3704

[38] A. Antonopoulos, "Bankruptcy problem in network sharing: Fundamentals, applications and challenges," *IEEE Wireless Commun.*, vol. 27, no. 4, pp. 81–87, Aug. 2020.

[39] A. Samanta and Z. Chang, "Adaptive service offloading for revenue maximization in mobile edge computing with delay-constraint," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 3864–3871, Apr. 2019.

[40] J. Huang, S. Li, and Y. Chen, "Revenue-optimal task scheduling and resource management for IoT batch jobs in mobile edge computing," *Peer-to-Peer Netw. Appl.*, vol. 13, pp. 1776–1787, Mar. 2020.

[41] 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.

[42] J. Fu, J. Hua, J. Wen, H. Chen, W. Lu, and J. Li, "Optimization of energy consumption in the MEC-assisted multi-user FD-SWIPT system," *IEEE Access*, vol. 8, pp. 21345–21354, 2020.

[43] Z. Zhou, J. Feng, Z. Chang, and X. Shen, "Energy-efficient edge computing service provisioning for vehicular networks: A consensus ADMM approach," *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 5087–5099, May 2019.

[44] A. Mukherjee, D. De, and D. G. Roy, "A power and latency aware cloudlet selection strategy for multi-cloudlet environment," *IEEE Trans. Cloud Comput.*, vol. 7, no. 1, pp. 141–154, Jan.–Mar. 2019.

[45] Y. Cui, W. He, C. Ni, C. Guo, and Z. Liu, "Energy-efficient resource allocation for cache-assisted mobile edge computing," in *Proc. IEEE Conf. Local Comput. Netw.*, Oct. 2017, pp. 640–648.

[46] J. Yao and N. Ansari, "QoS-aware fog resource provisioning and mobile device power control in IoT networks," *IEEE Trans. Netw. Service Manag.*, vol. 16, no. 1, pp. 167–175, Mar. 2019.

[47] J. Xiong, H. Guo, and J. Liu, "Task offloading in UAV-aided edge computing: Bit allocation and trajectory optimization," *IEEE Commun. Lett.*, vol. 23, no. 3, pp. 538–541, Mar. 2019.

[48] L. Chen, S. Zhou, and J. Xu, "Energy efficient mobile edge computing in dense cellular networks," in *Proc. IEEE Int. Conf. Commun.*, May 2017, pp. 1–6.

[49] S. Ranadheera, S. Maghsudi, and E. Hossain, "Computation offloading and activation of mobile edge computing servers: A minority game," *IEEE Wireless Commun. Lett.*, vol. 7, no. 5, pp. 688–691, Oct. 2018.

[50] Q. Wang, Q. Xie, N. Yu, H. Huang, and X. Jia, "Dynamic server switching for energy efficient mobile edge networks," in *Proc. IEEE Int. Conf. Commun.*, May 2019, pp. 1–6.

[51] A. Sundarrajan, M. Kasbekar, and R. K. Sitaraman, "Energy-efficient disk caching for content delivery," in *Proc. Int. Conf. Future Energy Syst.*, Jun. 2016, pp. 1–12.

[52] L. He, H. Xu, H. Wang, L. Huang, and J. Ma, "Task offloading in edge-clouds with budget constraints," *Lecture Notes in Computer Science*, vol. 11336. Cham, Switzerland: Springer, 2018, pp. 311–326.

[53] J. Zhou, J. Fan, J. Wang, and J. Jia, "Dynamic service deployment for budget-constrained mobile edge computing," *Concurrency Comput. Pract. Exp.*, vol. 31, no. 24, pp. 1–16, 2019.

[54] L. Chen and J. Xu, "Budget-constrained edge service provisioning with demand estimation via bandit learning," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 10, pp. 2364–2376, Oct. 2019.

[55] L. Chen, J. Wu, J. Zhang, H.-N. Dai, X. Long, and M. Yao, "Dependency-aware computation offloading for mobile edge computing with edge-cloud cooperation," *IEEE Trans. Cloud Comput.*, early access, Nov. 11, 2020, doi: 10.1109/TCC.2020.3037306.

[56] Y. Chen and L. Yen, "Distributed profitable deployment of network services to geo-distributed edge systems," in *Proc. Int. Asia-Pac. Netw. Oper. Manag. Symp. (APNOMS)*, Jun. 2016, pp. 1–12.

[57] Y. Sun, S. Zou, and J. Xu, "EMM: Energy-aware mobility management for mobile edge computing in ultra dense networks," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2637–2646, Nov. 2017.

[58] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1619–1630, Aug. 2018.

[59] H. Badri, T. Bahreini, D. Grosu, and K. Yang, "Risk-aware application placement in mobile edge computing: A learning-based optimization approach," *Proc. IEEE Int. Conf. Edge Comput.*, Oct. 2020, pp. 83–90.

[60] *Standard Performance Evaluation Corporation*. Accessed: Mar. 16, 2021. [Online]. Available: http://www.spec.org/power_ssj2008/results/

[61] G. Dahl and N. Foldnes, "LP based heuristics for the multiple knapsack problem with assignment restrictions," *Ann. Oper. Res.*, vol. 146, no. 1, pp. 91–104, 2006.

[62] M. Dawande, J. Kalagnanam, P. Keskinocak, R. Ravi, and F. S. Salman, "Approximation algorithms for the multiple knapsack problem with assignment restrictions," *J. Comb. Optim.*, vol. 4, pp. 171–186, Jun. 2000.

[63] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. Chichester, U.K.: Wiley, 1990.

[64] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 1989.

[65] V. Nguyen and Y. Tan, "Minimum convex cost flow problem," in *Proc. IEEE Int. Conf. Inf. Commun. Signal Process.*, Mar. 2003, pp. 1248–1252.

[66] A. Trivedi, L. Wang, H. Bal1, and A. Iosup, "Sharing and caring of data at the edge," in *Proc. USENIX Workshop Hot Topics Edge Comput.*, Jun. 2020, pp. 1–12.

[67] Y. Xie *et al.*, "Real-time prediction of docker container resource load based on a hybrid model of ARIMA and triple exponential smoothing," *IEEE Trans. Cloud Comput.*, early access, Apr. 22, 2020, doi: 10.1109/TCC.2020.2989631.
[68] J. Chavarriaga, A. Vega, E. Rosales, and H. Castro, "A Linux scheduler to limit the slowdown generated by volunteer computing systems," in *Proc. Int. Conf. Appl. Inform.*, Oct. 2020, pp. 149–164.
[69] H. Zhang, F. Guo, H. Ji, and C. Chu, "Combinational auction-based service provider selection in mobile edge computing networks," *IEEE Access*, vol. 5, pp. 13455–13464, 2020.
[70] *Shortest/ Path/ Faster/ Algorithm*. Accessed: Mar. 16, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Shortest_Path_Faster_Algorithm

Minseok Song (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer engineering from Seoul National University, Seoul, South Korea, in 1996, 1998, and 2004, respectively.

Since September 2005, he has been with the Department of Computer Engineering, Inha University, Incheon, South Korea, where he is currently a Professor. His research interests include embedded systems, multimedia, and IoT systems.

Yeongju Lee received the B.S., M.S., and Ph.D. degrees in computer engineering from Inha University, Incheon, South Korea, in 2011, 2013, and 2018, respectively.

He is currently working with FPT Software Korea, Seoul, South Korea. His current research interests include embedded software and ubiquitous computing.

Kyungmin Kim received the B.S. degree in computer engineering from Inha University, Incheon, South Korea, in 2020, where he is currently pursuing the M.S. degree.

His research interests include system software and cloud computing.