Received November 22, 2020, accepted November 28, 2020, date of publication December 2, 2020,
date of current version December 14, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3041872

Collaborative Task Scheduling for IoT-Assisted Edge Computing

YOUNGJIN KIM[1], CHIWON SONG[1], HYUCK HAN[2], HYUNGSOO JUNG[1], AND SOOYONG KANG[1], (Member, IEEE)
[1]Department of Computer Science, Hanyang University, Seoul 04763, South Korea
[2]Department of Computer Science, Dongduk Women's University, Seoul 02748, South Korea

Corresponding author: Sooyong Kang (sykang@hanyang.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) funded by the Korea Government (MSIP) under Grant 2019R1A2C1011009.

ABSTRACT The Internet of Things (IoT) is evolving rapidly and requires IoT devices to have more resources to meet the growing needs in diverse application domains. Despite increasing demands, modern IoT devices do not fully utilize somewhat over-provisioned computing resources. In this work, we introduce a new concept of *IoT-assisted Edge Computing* which makes use of consolidated idle resources in IoT devices for edge services through offloading edge tasks to nearby IoT devices. For the IoT-assisted edge computing be beneficent, two important conditions should be satisfied: 1) offloaded edge tasks to IoT devices do not hurt normal execution of local IoT tasks, and 2) computing resources in IoT devices should be effectively exploited to increase the throughput of edge services. To that end, we propose a collaborative task scheduling for IoT-assisted edge computing, in which an edge node determines where to offload edge tasks among participating IoT devices based on the offloaded execution time and energy consumption, and each IoT device determines when to execute the offloaded tasks considering local tasks execution. Experimental results show that the proposed scheme not only achieves near-optimal task throughput but also outperforms other scheduling algorithms in terms of deadline satisfaction ratio of time critical tasks, while guaranteeing deadlines of local tasks in IoT devices.

INDEX TERMS Edge service, IoT-assisted edge computing, IoT service, task scheduling.

I. INTRODUCTION

Recently, a new computing paradigm of Edge Computing has emerged to exploit the computing resources located at the edge of the network, i.e., *edge servers*, for processing all or part of user services. By using an edge server instead of the central server located in the data center, it is possible to overcome the hurdles of the network latency and the amount of traffic in the backbone network for time-critical (or -sensitive) cloud/IoT services. Considering that more and more end-user devices are being connected to the Internet via mobile networks, the effect of the edge computing is expected to become even stronger in the era of the 5G mobile network that offers much shorter latency and higher bandwidth than 4G/LTE network.

However, if we look at network edges, the edge computing also has a centralized architecture in that every nearby user's service request is delivered to the 'central' edge server. And since the computing power of edge servers is far weaker than that of cloud-based servers, edge servers will have non-trivial resource contention on limited computing resources between multiple competing tasks serving user requests. The excessive resource contention in an edge server undoubtedly increases the execution time of running tasks. Hence, an efficient task scheduling algorithm must be proposed to deal with highly over-loaded situations in an edge server. For this purpose, researchers in our community have made efforts to address this issue and developed novel task scheduling algorithms that leverage resources in cloud servers [1]–[4].

On the other hand, it is well recognized that IoT devices introduced recently have more computing power, and they become capable enough to collect and process data in the device, without sending it to a server. For example, Qualcomm announced their plan to supply high-end CPUs for IoT devices that enables IoT devices to provide various real-time AI services under the slogan 'On-device AI Ubiquitous' [5], and NVIDIA is preparing for future IoT services including

The associate editor coordinating the review of this manuscript and approving it for publication was Ligang He.

vulnerable road user discovery (VRU) service via supplying their GPUs to smart cars [6]. These high-end IoT devices are already equipped with multicore CPU, multi-gigabytes of memory and 5G communication module to execute complex, computation-intensive tasks. In this work, we aim to find a way to utilize the unused or idle computing resources in IoT devices, that are available when the devices are underutilized. By collectively exploiting over-provisioned idle resources as an extra resource pool for providing edge services, we can substantially increase the effectiveness of the edge computing by offloading some tasks in an edge server (i.e., edge tasks) to nearby IoT devices. In this paper, we call this paradigm as '*IoT-assisted Edge Computing*'.

Similar concepts with our IoT-assisted edge computing have been introduced recently, including *ad-hoc cloud computing* and *device-enhanced MEC*. Ad-hoc cloud computing is a generic concept of building a virtual cloud infrastructure exploiting resources from existing sporadically available, non-exclusive, and unreliable infrastructures [7]. It harvests resources from underutilized PCs or servers with enough resources to execute tasks on the virtual machines, challenging to apply to relatively resource-poor IoT devices that need fine-grained resource control. Device-enhanced MEC is a concept of exploiting resources in both MEC servers and end devices (e.g., mobile phones) to execute end devices' tasks [8] collaboratively. While it uses D2D communications to exploit resources in remote user devices, in IoT-assisted edge computing, the edge server exploits resources in local IoT devices under its governance without involving D2D communications among end devices. More importantly, both of the previous concepts aim to *provide* resources to end-users; hence tasks are offloaded 'from user devices to infrastructure'. On the other hand, IoT-assisted edge computing aims to *exploit* resources in end devices (i.e., IoT) to execute edge tasks, and therefore, tasks are offloaded 'from the edge server to end devices'.

Edge tasks are dynamically generated when a user calls for a service that the edge server provides, and multiple tasks can be generated for a single service request. By offloading a portion of edge tasks to IoT devices, the resource contention in an edge server can be reduced, and therefore IoT-assisted edge computing can achieve better task throughput than ordinary edge computing. To that end, an edge server needs to have a task scheduler that determines which tasks will be offloaded to which IoT device. One simple and efficient approach is to assign a task based on data proximity. However, we claim that there exist four important issues that must be considered in designing a more efficient and general task scheduling scheme.

First, the performance gap between IoT devices and an edge server, or between different IoT devices, needs to be considered in designing a new scheduling scheme. Otherwise, the overall user service time can become unacceptable due to a slow device that executes large tasks for the service. Hence, both the resource requirement of each task and the real-time resource availability in each IoT device should be taken into consideration together in scheduling tasks.

Second, there can be time-critical services (e.g., VRU service) whose tasks should be completed within specified deadlines. Hence, when multiple users request time-critical (or -sensitive) and best-effort services at the same time to the edge server, we should not only give execution preference to tasks of the time-critical services to meet their deadlines but also minimize the entire execution time of all tasks to maximize the throughput of the edge computing.

Third, execution of offloaded tasks inevitably consumes extra energy in IoT devices. Excessive task offloading may drain an IoT device of its energy and impede the execution of IoT device's own tasks (in this paper, we call it 'local' tasks) due to the energy deficiency; but the execution of local tasks is indeed the main role of IoT devices. Therefore, the task scheduler in an edge server should be able to anticipate the increased energy consumption due to forthcoming offloaded tasks in each IoT device, and to proactively decide whether or not to offload tasks for the proper execution of local tasks in IoT devices.

Finally, task offloading should be non-invasive in that offloaded tasks should not delay the execution of local tasks in IoT devices. This constraint makes any realistic task scheduling scheme for IoT-assisted edge computing cannot be optimal, since local tasks can be spontaneously and intermittently executed. For example, a face recognition task in a smart door bell is executed only when an object (probably, a visitor) is detected by its camera. Hence, an edge server cannot generate the optimal task scheduling plan which presumably but accurately incorporates unpredictable local tasks in IoT devices. This is why the task schedule (generated by an edge server) to be updated by each IoT device who has the entire control of its own tasks.

In this paper, we propose a *collaborative* task scheduling between edge server and IoT devices that addresses the above issues. In a nutshell, an edge server only determines tasks that will be offloaded to each participating IoT device, and each IoT device determines the execution schedule of the offloaded tasks to itself incorporating its local tasks. When determining tasks that will be offloaded to each IoT node, the edge server considers both the overall offloaded execution time and energy consumption. Each IoT device, instead of generating a static and serial task schedule, uses a weight-based resource allocation (like Linux's `cgroup` [9]) to dynamically determine the task execution schedule considering both the deadline and execution time of each task. Experimental results show that our collaborative task scheduling not only achieves near-optimal task throughput but also outperforms legacy scheduling schemes in terms of deadline satisfaction ratio of time critical tasks, while guaranteeing deadlines of local tasks in IoT devices.

The contribution of this paper is three fold.

1) We establish communication, computation and energy consumption models and define formal task assignment

and scheduling problems in IoT-assisted edge computing.

2) We propose a collaborative task scheduling scheme that considers unique characteristics of the IoT-assisted edge computing.
3) We propose two heuristic algorithms that realize our collaborative task scheduling, a completion time-based task assignment algorithm and a hierarchical weight allocation algorithm, each of which solves task assignment and scheduling problems, respectively.

The rest of this paper is organized as follows. Section II presents the related work. In Section III, we establish models for IoT-assisted edge computing, and we propose our collaborative task scheduling scheme in Section IV. In Section V, we evaluate the performance of the proposed scheme and finally, in Section VI, we conclude this paper.

II. RELATED WORK AND MOTIVATION

A. TASK OFFLOADING IN EDGE COMPUTING

There have been many studies that use cloud computing to solve problems caused by low performance or lack of resources of mobile devices [10], [27]–[31]. The basic idea of these studies is to offload parts of computationally-heavy tasks on mobile devices to the resource-rich cloud and to perform offloaded tasks within the cloud. A good example is the work presented by Clark *et al.* [32], that proposed a concept of a virtual machine (VM)-based remote execution scheme in which the live migration of virtual machines enables an entire OS including all running applications to be moved to other virtual machine. Based on this technique, CloneCloud [27] and Cloudlet [28] proposed methods to offload computation to cloud by executing tasks of mobile devices on remote VMs without programmers' efforts. These studies focus on cloud offloading to overcome the low performance rather than to increase the energy-efficiency of mobile devices. For the energy-efficiency of mobile devices, offloading methods have been proposed in [29]–[31], [33], and a hybrid strategy considering performance and energy consumption was also proposed in [10]. However, the explosion of mobile and IoT devices causes massive data traffic in backbone networks, and this incurs numerous performance problems due to the long latency of cloud offloading [34].

To address the issues, a new concept of edge computing [35] has been proposed to offload tasks to edge servers that can bypass the slow backbone networks. Many research projects related to edge computing have focused on both performance and energy consumption for energy-poor IoT and mobile devices. TAPDEF [23] uses Unmanned Aerial Vehicles (UAV) such as drones for IoT devices that do not have direct access to cloud, and IoT devices can distribute their tasks to cloud through UAV devices. For this purpose, an optimized algorithm that minimizes energy consumption and execution delay is also proposed. RT-EEM [13] proposes a cooperative resource allocation framework to enhance overall energy efficiency by considering the distance between IoT devices and the edge server. For this end, when an IoT device is hard to access the edge server due to the long distance between the two nodes, the IoT device offloads its tasks to the edge server through other IoT devices near the edge server. ACOA [12] establishes task route paths with minimal energy consumption when distributing work from IoT devices to edge servers on the 5G network. For this, the artificial fish swarm algorithm (AFSA) is developed to decide which femto relay base stations (FRSs) of 5G networks are passed through for the task distribution. ISA-TOTPA [14] suggests an algorithm in which an optimal CPU frequency that could lead to minimal energy consumption is calculated and tasks are offloaded to an appropriate edge server based on the calculation. GMS [15] suggests the greedy-based algorithms for distributing tasks from a mobile device (MD) to the mobile edge cloud consisting of multiple energy harvesting wireless access points (APs).

B. TASK SCHEDULING IN EDGE COMPUTING

As the use of IoT and mobile devices becomes prevalent in the edge computing community, the importance of scheduling techniques in edge servers has also increased, and various methods have been proposed.

Some of studies aimed at optimizing energy consumption. In RTA [36], energy allocation and task scheduling techniques are proposed when energy is transferred from the edge server or the access point to energy harvesting IoT devices. For energy-efficiency, task scheduling and energy allocation are performed on IoT devices and edge servers, respectively. In [11], a joint allocation algorithm is proposed for task scheduling among many available ones, with heterogeneous latency requirements being imposed.

In addition to these studies, there have been several research projects minimizing the execution delay. The UL-DL scheduler [16] minimizes the execution time of bursty tasks using the queuing theory. In LDP-JCTO [19], data partitioning techniques are designed for efficient distributed data processing with heterogeneous IoT devices, which reduces the execution time for data processing. A study at IHRA [18] proposed a task scheduling technique intended to minimize the overall execution time by considering both cloud and edge servers together when IoT devices are initialized and have local tasks offloaded to the servers. In particular, it develops feedback functions for detecting the status of edge servers since the resource contention problem arises in hotspot edge servers. In MSGA [17], the data location and the congestion in networks are considered when transferring tasks from IoT devices to edge servers. If networks are congested, searching a detour mechanism is triggered based on the past history of flow states.

Some of studies aimed at minimizing both execution time and energy consumption by devising novel resource allocation and scheduling techniques. In [21], they propose an optimal resource allocation scheme for mobile edge computation offloading system based on time-division multiple access (TDMA) and orthogonal frequency-division multiple access (OFDMA). In [22] authors propose a resource

TABLE 1. Related work to task offloading and scheduling in Edge computing. (D/L: Deadline).

Prior work	Objective	Offload from	Offload to	Decision Maker	when offloading to IoT devices: Local (IoT) Tasks Consideration	when offloading to IoT devices: Per-device Energy Consideration	when offloading to IoT devices: Scheduling in IoT[§]
Guo et al. [10]	Energy	Mobile	Cloud	Mobile	-	-	-
Yu et al. [11]	Energy	IoT	Edge	IoT	-	-	-
Yang et al. [12]	Energy	IoT	Edge	IoT	-	-	-
Ji et al. [13]	Energy	IoT	Edge	IoT	-	-	-
Guo et al. [14]	Energy	IoT	Edge	IoT	-	-	-
Chen et al. [15]	Energy	Mobile	Edge	Mobile	-	-	-
Molina et al. [16]	Delay	IoT	Edge	IoT	-	-	-
Sahni et al. [17]	Delay	IoT	Edge	IoT	-	-	-
Ning et al. [18]	Delay	IoT	Edge	IoT	-	-	-
Ju et al. [19]	Delay	Edge	IoT	Edge & IoT	No	Yes	No
Zhang et al. [20]	Payoff	IoT	Edge	IoT	-	-	-
You et al. [21]	Energy & Delay	IoT	Edge	Edge	-	-	-
Dinh et al. [22]	Energy & Delay	IoT	Edge	IoT	-	-	-
Wei et al. [23]	Energy & Delay	IoT	Cloud	UAV	-	-	-
Gao et al. [24]	Energy & Delay	IoT	IoT or Edge	Edge	No	Yes	No
Zhang et al. [25], [26]	Delay & Payoff	Edge	IoT	Edge & IoT	No	Yes	No
This work	D/L & Throughput	Edge	IoT or Cloud	Edge	Yes	Yes	Yes

[§] 'Scheduling in IoT' also represents whether or not a new task can be offloaded to an IoT device before the previously offloaded task finishes.

allocation framework that optimizes both energy consumption and execution time when assigning tasks to multiple edge servers as well as a single edge server. In [24], they choose an appropriate computation offloading mode with social awareness-aided network resource assignment.

Other studies tried to achieve both the minimum execution time of applications and the maximum compensation of devices or server providers when processing data using IoT devices and edge servers. To that end, there have been game-theoretic approaches [25], [26] since finding the optimal result satisfying both objectives is an NP-hard problem. In EWBS [20], an efficient wholesale and buyback algorithm is designed to maximize the profit of edge sever providers.

C. MOTIVATION

Research communities have long studied task offloading and scheduling problems in edge computing. However, we found three crucial hurdles for using them in IoT-assisted edge computing through an in-depth analysis of prior proposals. First, most of the prior schemes assumed that tasks are offloaded from end (IoT/mobile) devices to edge/cloud servers to either save energy in end devices or enable resource-poor end devices to execute heavy applications with the help of resource-rich servers. Hence, it is hard to use such schemes in IoT-assisted edge computing environments where tasks are offloaded from edge servers to IoT devices (i.e., in the reverse direction) to increase the service throughput in edge servers.

Second, even in prior studies that consider offloading tasks to IoT devices [19], [24]–[26], they did not consider offloading/executing *multiple* tasks simultaneously to/in end devices. So a specific scheduling scheme for multiple tasks in end devices was not necessary for those proposals.

Third, since local tasks' safe execution is the leading role of IoT devices, it is of utmost importance to protect local tasks in the presence of offloaded external tasks. To execute offloaded tasks efficiently without delaying local tasks in IoT devices, we require an IoT device to have its task scheduling scheme since the IoT device itself is the only entity with full control of the local tasks. However, to the best of our knowledge, there is no prior work that provides a scheduling scheme for IoT devices that schedules multiple tasks, including offloaded and local tasks. We believe that this is due to the assumption that prior researchers accepted; considering IoT devices' insufficient resource capacity, it is natural to offload only one task to an IoT device when it is idle. Under such an assumption, IoT devices do not need to have a specific scheduling scheme for multiple tasks. However, considering ever-increasing (and sometimes over-provisioned) resource capacity in IoT devices, our community must consider offloading multiple tasks to an IoT device. Moreover, since simultaneous execution of multiple offloaded tasks in an IoT device undoubtedly affects the safe execution of intermittently and unpredictably generated local IoT tasks, we need a carefully designed task scheduling scheme that prioritizes local tasks while maximizing the total task throughput.

The above observations motivated this work. This paper's proposed scheme addresses such issues and considers the growing demand for real-time edge services that generate time-critical tasks with their execution deadlines.

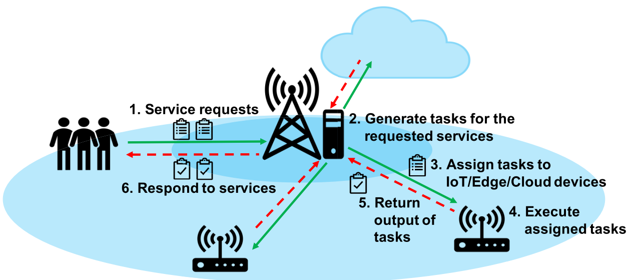

1. Service requests
2. Generate tasks for the requested services
3. Assign tasks to IoT/Edge/Cloud devices
4. Execute assigned tasks
5. Return output of tasks
6. Respond to services

FIGURE 1. System model of the IoT-assisted edge computing.

III. MODELS AND PROBLEMS

This section presents communication, computation and energy consumption models in the IoT-assisted edge computing. We first illustrate the overall system model in Figure 1.

Three kinds of devices (IoT devices, edge and cloud servers) participate in the IoT-assisted edge computing. IoT devices are usually resource-poor in that they have limited computational power, and it is directly connected via wireless networks to the Base Station (BS) where edge servers are located. We assume that there is no way for IoT devices to do direct communication between devices, since IoT devices do not recognize each other [37]–[39].

A user request is sent to an edge server either directly or indirectly via cloud servers. The edge server upon receiving the request spawns tasks for handling each requested service and dispatches the tasks to IoT devices, cloud servers or itself for execution. The execution results are transferred to the edge server who will send a response to the user.

TABLE 2. **Notations.**

Notation	Description
$\mathcal{M}$	Set of all devices in IoT-assisted edge computing
ϵ, c	Edge and Cloud servers, respectively ($\epsilon, c \in \mathcal{M}$)
d	IoT device ($d \in \mathcal{M}$)
f_A	CPU frequency of device A ($\in \mathcal{M}$)
$\mathcal{T}$	Set of all tasks to be scheduled
$\mathcal{T}_A$	Set of tasks offloaded to device A
t_i	Task i ($t_i \in \mathcal{T}$)
C_i	Required CPU cycles to complete t_i
S_i	Size of the executable image of t_i
I_i, O_i	Input and Output data sizes, respectively, for t_i
D_i	Deadline of t_i
C_A^{avg}	Average CPU cycles consumed in a second by background tasks in device A
$r_{\epsilon,d}$	Wireless network bandwidth from ϵ to d
$T_{\epsilon,A}^{i,exe}$	Transfer time of the executable image of t_i from ϵ to A ($\in \mathcal{M}$)
$T_{A,B}^{i,in}$	Transfer time of input data of t_i from A to B. If $A = B$ then $T_{A,B}^{i,in} = 0$.
$T_{A,\epsilon}^{i,out}$	Transfer time of output data of t_i from A to ϵ
$T_{\epsilon,A}^{i}$	Total communication time for the offloaded execution of t_i to A ($\in \mathcal{M}$)
$T_A^{i,cpu}$	Execution time for completing t_i in A ($\in \mathcal{M}$)
$E_d^{i,rx}$	Energy consumption in d for receiving executable image and input data of t_i from ϵ
$E_d^{i,tx}$	Energy consumption in d for transmitting output data of t_i to ϵ
$E_d^{i,cpu}$	CPU energy consumption for executing t_i in d
E_d^i	Total energy consumption for the offloaded execution of t_i to d
E_d	Current energy level of d
E_d^θ	Minimum energy level threshold of d
$\lambda_{i,A}$	Assignment indicator of t_i to A ($\lambda_{i,A} \in \{0,1\}$)
$\lambda_{i,A}^I$	Indicator of the existence of the input data of t_i in A ($\lambda_{i,A}^I \in \{0,1\}$, $A \in \mathcal{M}$)

Table 2 lists notations used in our models. A task t_i ($\in \mathcal{T}$) is independently executed in a single device (IoT, edge server or cloud server) and represented as a quintuple, $t_i = <C_i, S_i, I_i, O_i, D_i>$, in which each element is determined in advance via task profiling. Whenever a request for a service arrives to the edge server, it generates tasks for the service and determines which tasks will be offloaded to which device for execution considering task and data transfer time, task execution time and the amount of energy consumption, each of which is estimated based on the communication, computation and energy consumption models, respectively.

A. COMMUNICATION TIME MODEL

Task offloading from the edge server to other devices requires task and data transfer. The communication overhead between edge and other device is one of key factors in determining task assignment. Since IoT devices are directly connected to the base station via wireless networks, we can model their communication overhead using Shannon's law [40]. The wireless network throughput from the edge server to an IoT device ($r_{\epsilon,d}$) can be modeled as

$$r_{\epsilon,d} = B_c log_2(1 + \frac{g_{\epsilon,d} p_\epsilon^{tx}}{\sigma}) \tag{1}$$

where B_c, $g_{\epsilon,d}$, p_ϵ^{tx} and σ represent channel bandwidth, channel gain, wireless transmit power in the edge server, and power of the noise and interference, respectively. The network throughput of the reverse direction ($r_{d,\epsilon}$) can be calculated similarly. Using (1), we can get each communication time for executable image, input data and output data of a task t_i between the edge server and an IoT device d as:

$$T_{\epsilon,d}^{i,exe} = \frac{S_i}{r_{\epsilon,d}}, \quad T_{\epsilon,d}^{i,in} = \frac{I_i}{r_{\epsilon,d}}, \quad T_{d,\epsilon}^{i,in} = \frac{I_i}{r_{d,\epsilon}}, \quad T_{d,\epsilon}^{i,out} = \frac{O_i}{r_{d,\epsilon}}$$

A task can also be offloaded to the cloud server. Since the edge and cloud servers are connected through a wired network (i.e., the Internet), we cannot use (1). Besides the data size and network bandwidth, the communication time in this case depends also on the hop count between two servers and the queuing delay in each intermediate router, among others. For simplicity, we define a delay parameter, δ, that represents aggregate communication delay induced by such factors other than the amount of data and the network bandwidth. Then, a communication time for an executable image, input and output data between edge and cloud servers can be expressed as:

$$T_{\epsilon,c}^{i,exe} = \frac{S_i}{B} + \delta, \quad T_{\epsilon,c}^{i,in} = T_{c,\epsilon}^{i,in} = \frac{I_i}{B} + \delta, \quad T_{c,\epsilon}^{i,out} = \frac{O_i}{B} + \delta$$

where B denotes the network bandwidth between them.

The total communication time for task offloading from an edge server to device A ($\neq\epsilon$) is affected by the current location of input data. First, if input data is in A then we do not need to transfer data. Second, if it is stored in the edge server but not in A, then we should transfer data from the edge server to A. Finally, if data is in other non-edge device, not A, then input data should be transferred from the device to the edge server first, and then from the edge server to A. To take the location of input data into consideration in designing our model, we define a binary variable $\lambda_{i,A}^I$ which is 1 when the input data of t_i is in device A ($\in \mathcal{M}$), and is 0 otherwise. Then, using $\lambda_{i,A}^I$, we can model the total communication time for the offloaded execution of t_i from the edge server to a non-edge device A as:

$$T_{\epsilon,A}^i = T_{\epsilon,A}^{i,exe} + (1 - \lambda_{i,A}^I)(T_{\hat{A},\epsilon}^{i,in} + T_{\epsilon,A}^{i,in}) + T_{A,\epsilon}^{i,out}, \quad \text{if } A \neq \epsilon \tag{2}$$

where $\hat{A}$ denotes a device that has the input data. It is notable that if input data is in an edge server (i.e., $\hat{A} = \epsilon$), the term $T^{i,in}_{\hat{A},\epsilon}$ becomes zero.

A task can be executed in the edge server itself without being offloaded. In that case ($A = \epsilon$), the communication time includes only the time to transmit input data if it is not in an edge server, i.e., $T^{i,exe}_{\epsilon,A} = T^{i,in}_{\epsilon,A} = T^{i,out}_{A,\epsilon} = 0$ in (2)

B. COMPUTATION TIME MODEL

Task execution time is another important factor in task scheduling. We assume that each task has already been profiled and an edge server knows the CPU cycles required for completing a given task. Then the execution time of t_i in a device A ($A \in \mathcal{M}$) can be modeled as

$$T^{i,cpu}_{A} = \frac{C_i}{f_A - C^{avg}_{A}}, \tag{3}$$

where C^{avg}_{A} is the *average* CPU cycles consumed in a second by background tasks (i.e., ever-running tasks not for user-requested services) in device A, including *ever-running* local tasks in IoT devices, OS kernel and system management tasks, among others.

We assume that input/output data is stored in memory (not storage) so that task execution does not incur any storage I/O, and this assumption is realized through the use of in-memory file systems (e.g., *tmpfs*) for data storage. Hence, we can safely disregard the necessity of considering I/O overhead in our computation time model; because modeling I/O overhead is generally complicated and hardly accurate.

C. ENERGY CONSUMPTION MODEL

Being different from edge and cloud servers, large types of IoT devices either use their own batteries or harvest energy for operations. Hence, their energy consumption due to the offloaded tasks becomes matter in IoT-assisted edge computing. To control the 'increased' energy consumption due to the offloaded tasks in IoT devices, we need to precisely estimate and incorporate it to task assignment.

The CPU power consumption in a device d can be modeled as $P_d = kf_d^3$ [31], where k and f_d represent the coefficient depending on the CPU architecture and the CPU frequency of d, respectively. Ignoring the power consumption due to the CPU scheduling itself, the CPU energy consumption to execute t_i in an IoT device d can be modeled as

$$E^{i,cpu}_{d} = P_d \times \frac{C_i}{f_d} = C_i k f_d^2. \tag{4}$$

For each offloaded task to an IoT device, the executable image of the task and its input data (if necessary) are transferred from an edge server to an IoT device, and the output of the task is also transferred in the opposite direction. Such communications naturally consume energy in IoT devices, and can be modeled as

$$E^{i,rx}_{d} = p^{rx}_{d} \times \frac{S_i + (1 - \lambda^I_{i,d})I_i}{r_{\epsilon,d}}, \tag{5}$$

$$E^{i,tx}_{d} = p^{tx}_{d} \times \frac{O_i}{r_{d,\epsilon}}, \tag{6}$$

where p^{tx}_{d} and p^{rx}_{d} denote wireless transmit and receive power, respectively, in an IoT device d.

Putting all together, the total energy consumption in d (E^i_d) for the offloaded execution of t_i to d becomes

$$E^i_d = E^{i,rx}_{d} + E^{i,cpu}_{d} + E^{i,tx}_{d}. \tag{7}$$

D. PROBLEM FORMULATION

In this section, we present formal definitions of task assignment and scheduling problems in IoT-assisted edge computing. Briefly, the task assignment problem is to find the optimal task distribution across all devices that minimizes the completion time of all tasks. And the task scheduling problem is to find the optimal schedule of task execution in each device that maximizes the number of tasks that meet their respective deadlines.

There are a few constraints that must be satisfied while finding the optimal task distribution. First, a task is assigned to only one device/server. Formally,

$$\forall t_i \in \mathcal{T}, \quad \sum_{A \in \mathcal{M}} \lambda_{i,A} = 1 \tag{8}$$

where $\lambda_{i,A}$ is a binary variable that indicates whether or not a task t_i is assigned to device A, specifically, it is 1 when t_i is assigned to A and 0 otherwise.

Second, input data for each task may exist in one or more devices. If a task does not need any input data (for example, a mathematical function that use only its parameters) then we can regard it as a task having its input data in every device. Hence, we have the following constraint:

$$\forall t_i \in \mathcal{T}, \quad \sum_{A \in \mathcal{M}} \lambda^I_{i,A} \geq 1 \tag{9}$$

Third, every IoT device may have its own minimum energy level requirement (E^θ_d) determined by the energy consumption required for executing its local tasks. Hence, the energy level of an IoT device, after completing all the offloaded tasks, should never drop below the minimum energy level threshold of the device:

$$\forall d, \quad E_d - \sum_{t_i \in \mathcal{T}} \lambda_{i,d} E^i_d \geq E^\theta_d \tag{10}$$

Now, we define the task assignment problem.

Definition 1 (Task Assignment Problem): For all tasks, find devices in which every task will be executed such that

$$\textit{minimize}: \max_{A \in \mathcal{M}} \sum_{t_i \in \mathcal{T}} \lambda_{i,A}(T^{i,cpu}_{A} + T^i_{\epsilon,A})$$
$$\text{subject to (8), (9) and (10).}$$

Given the constraint (8), the number of ways to assign all the tasks across all devices is $|\mathcal{M}|^{|\mathcal{T}|}$, which is the size of search space to solve the task assignment problem. Considering that a large number of IoT devices and end-user devices (including mobile phones and vehicles) are expected to be connected to an edge server for upcoming edge computing services, we need a heuristic solution to the problem.

For task scheduling in each device, we first define a notation, $\mathcal{T}_A$, which denotes a set of all tasks assigned to a device A. Assuming that $|\mathcal{T}_A| = n$ $(\leq |\mathcal{T}|)$, let $\mathcal{S}_A = (t_0, t_1, \cdots, t_{n-1})$ be a sequence of task executions in A, and $T_A^{Fin_i}$ be the very time when t_i finished its execution in A. At the time device A generates $\mathcal{S}_A$, it is safe to assume that all executable images for n tasks have already arrived to A. But since their input data can still be missing in A, the communication time for input and output data should be reflected in estimating the finish time of a task. Hence, we can define the finish time of a task t_i in A as

$$T_A^{Fin_i} = \begin{cases} T_A^{Fin_{i-1}} + T_A^{i,cpu} + (T_{\epsilon,A}^{i} - T_{\epsilon,A}^{i,exe}) & \text{if } i \geq 1 \\ T_{cur} + T_A^{i,cpu} + (T_{\epsilon,A}^{i} - T_{\epsilon,A}^{i,exe}) & \text{if } i = 0, \end{cases}$$

where T_{cur} is the current time.

A task t_i is said to satisfy its deadline (D_i) if its finish time is not later than its deadline. Formally, letting $H_{i,A}$ indicate whether or not t_i meets its deadline in its assigned device A,

$$H_{i,A} = \begin{cases} 1 & \text{if } T_A^{Fin_i} \leq D_i \\ 0 & \text{otherwise.} \end{cases}$$

Now, we formally define the task scheduling problem.

Definition 2 (Task scheduling Problem): Given a set of assigned tasks to each device, find a sequence of task executions in each device such that

$$maximize: \sum_{A \in \mathcal{M}} \sum_{t_i \in \mathcal{T}_A} H_{i,A}$$

The large search space, $\prod_{A \in \mathcal{M}}(|\mathcal{T}_A|!)$, of the task scheduling problem also necessitates a heuristic algorithm.

Before giving heuristic solutions for task assignment and scheduling problems, it is worth investigating to see the orthogonality of the two problems. In other words, given a set of tasks, is it always possible to find a task distribution that not only minimizes the overall task execution time but also guarantees that a larger number of tasks can meet their deadlines than other task distributions? A simple count-example answers the question: Assume that there are two tasks (t_1 and t_2) whose deadlines are 5 and 10 seconds after the current time, respectively, and two devices A and B. The execution times of t_1 including communication overhead, $T_A^{1,cpu} + T_{\epsilon,A}^{1}$, in devices A and B are 5 and 6 seconds, respectively, and t_2 takes 6 and 9 seconds, respectively. The optimal task distribution is to assign t_1 and t_2 to devices B and A, respectively, in which case the overall execution time is minimized (6 seconds). In that case, t_1 cannot meet its deadline. However, if we assign t_1 and t_2 to devices A and B, respectively, both tasks meet their deadlines although the overall execution time (9 seconds) is not minimal. Hence, there is no solution that optimizes both the overall execution time and the number of deadline-met tasks, which guides us to develop a heuristic task assignment and scheduling solution that achieves both goals to a reasonable level.

IV. COLLABORATIVE TASK SCHEDULING

The impossibility in finding a solution that can optimize both the overall execution time and the number of deadline-met tasks is due also to the unpredictability of local task executions in IoT devices. Apart from their complexity, the problems in Definitions 1 and 2 need to be solved at the same time with the same given information. However, there is no way for an edge server to exactly consider the spontaneous execution of local tasks in IoT devices to resolve task assignment and scheduling problems. Hence, in this work, we propose a collaborative task scheduling scheme in which the edge server only addresses the task assignment problem considering task completion time, and each IoT device takes the responsibility of finding the execution schedule of assigned tasks with its local tasks being considered.

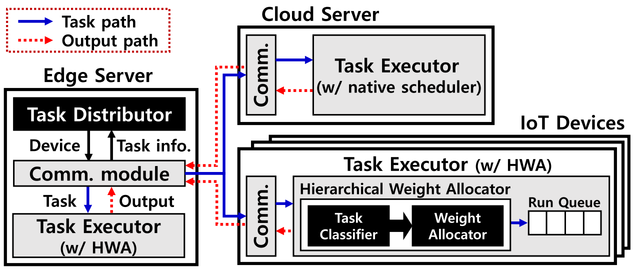


FIGURE 2. Software architecture for the collaborative task scheduling.

Figure 2 shows the software architecture for our collaborative task scheduling. It consists of a communication module and a task executor in each device and a task distributor in the edge server. The communication module takes the role of sending/receiving tasks and data among devices, and the task executor runs tasks. The task distributor chooses a device in which each task will be executed. Upon receiving a new service request from a user, an edge server sends tasks for the service to devices determined by the task distributor. Each device then executes received tasks and send their output back to the edge server. The edge server aggregates the outputs to respond to the service request. The core of the collaborative task scheduling scheme is two heuristic algorithms, Completion time-based Task Assignment (CTA) and Hierarchical Weight Allocation (HWA), used in the task distributor and the task executor, respectively. The CTA algorithm, whenever a new task is generated, selects the target device (i.e., a device in which the task will be executed) and sends it to the device as soon as possible, unlike prior studies ([17], [18], [20], [24], [41]) in which tasks are aggregated during an epoch and distributed to devices simultaneously at the end of the epoch. This real time task distribution is particularly effective to time-critical tasks. The HWA algorithm indirectly obtains the execution schedule of tasks by exploiting weight-based resource sharing mechanism in modern operating systems (for example, `cgroup` in Linux). Upon receiving a new task or whenever a local task needs to be executed, the HWA algorithm reassigns the weight of each task (either existing or new) not only to maximize the deadline-met tasks but also to guarantee the timely execution of the local tasks.

A. COMPLETION TIME-BASED TASK ASSIGNMENT

The objective of the CTA algorithm is to make the completion time of each task as early as possible in order to increase the probability that each task meets its deadline. As a heuristic approach to this objective, CTA assigns a task to a device where the task is expected to finish earliest. To this end, we estimate the expected finish time of a task in each device by summing the estimated execution time of the task and the time for completing all remaining tasks in a device.

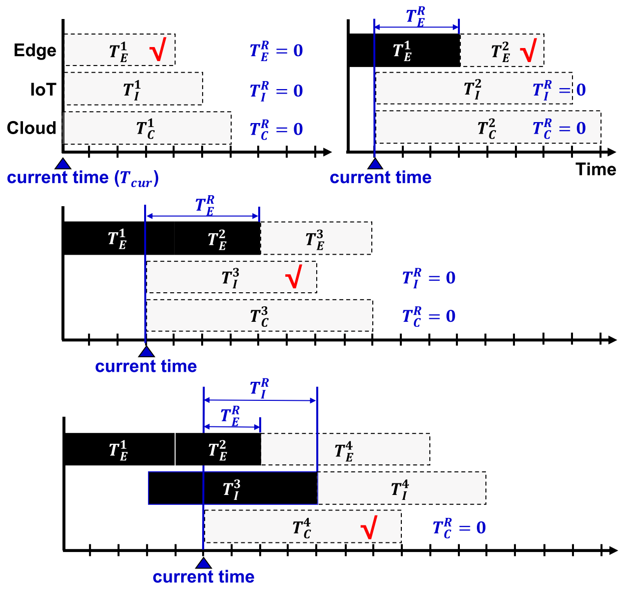


FIGURE 3. Completion time-based Task assignment: T_A^i in each rectangle represents the estimated execution time of t_i in device A. The check mark indicates the determined target device by CTA algorithm. And black rectangles represent assigned tasks to their target devices.

Let T_A^i be the total execution time (including communication delay) for a task t_i in a device A (i.e., $T_A^i = T_A^{i,cpu} + T_{\epsilon,A}^i$). And let T_A^R be the *remaining time* to complete all previously assigned but unfinished tasks in A. Then we can estimate the expected finish time of a new task t_i when it is executed in device A as $T_A^{Fin_i} = T_A^R + T_A^i$. The CTA algorithm selects the target device such that the expected finish time can be minimized. Figure 3 shows an example of the completion time-based task assignment, assuming three devices (Edge server: E, IoT device: I and Cloud server: C) and four tasks, t_1, t_2, t_3 and t_4, generated at times 0, 1, 3 and 5, respectively. Based on the estimated finish time of each task, the CTA algorithm finds destination devices—E, E, I and C— for the four tasks. Although not represented in this example, the CTA algorithm also considers the remaining energy of each IoT device when finding a target device. If the energy level of an IoT device d after executing t_i is estimated to be less than the predefined minimum energy level threshold of d, i.e., $E_d - E_d^i < E_d^\theta$, then the device is excluded from the candidate target devices to protect its local tasks from energy deficiency. To that end, each IoT device periodically reports its current energy level to the edge server which maintains energy information (E_d, E_d^θ) of each IoT device.

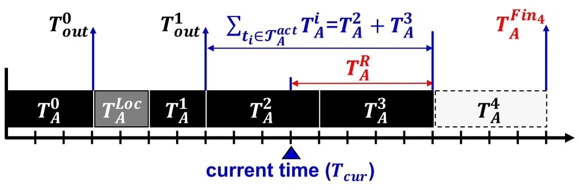


FIGURE 4. Remaining time estimation in CTA algorithm: T_A^{Loc} denotes the execution time of the intermittently executed local task in A, which is unpredictable in the edge server.

The noticeable challenge in implementing CTA algorithm is to estimate the remaining time (T_A^R) in each device, since edge servers have no prior knowledge on the current state of a device. To address this, the CTA algorithm exploits the most recent output creation time. When a task finishes its execution, it produces the final output. Then the communication module sends the output with its creation time to the edge server. On receiving the output, an edge server acknowledges the completion of the task. The creation time of the latest output arrived from a device provides us meaningful information to estimate the remaining time in the device. Figure 4 shows how to estimate the remaining time in device A (T_A^R) and the expected finish time of a new task (t_4) in A ($T_A^{Fin_4}$), assuming that four tasks ($t_0, \cdots, t_3$) have already been assigned to A. In this device, t_0 and t_1 have finished executions and their final output has been generated at time T_{out}^0 and T_{out}^1, respectively, and has arrived to the edge server. Since the output of t_2 and t_3 has not arrived yet, we assume that they are unfinished and one of them has started execution right after finishing the previous task (i.e., at time T_{out}^1). Then, the remaining time to finish both of them becomes $T_A^2 + T_A^3 - (T_{cur} - T_{out}^1)$, and the estimated finish time of t_4 in A becomes $T_A^{Fin_4} = T_A^2 + T_A^3 + T_A^4 - (T_{cur} - T_{out}^1)$. Equation (11) formalizes the remaining time estimation.

$$T_A^R = \max\{0, \sum_{t_i \in \mathcal{T}_A^{act}} T_A^i - (T_{cur} - T_{out}^{last})\} \tag{11}$$

In this formula, $\mathcal{T}_A^{act}$ denotes the set of active tasks in A whose outputs have not arrived yet and T_{out}^{last} denotes the creation time of the most recent output from A.

This way of estimating the remaining time is likely to be inaccurate if intermittent local tasks in IoT devices are executed after the estimation. However, the errors due to such tasks are not accumulated over time since the most recent output creation time accurately reflects the execution time of the preceding local tasks, as shown in Figure 4.

The CTA algorithm chooses a target device for a single task when the task is created with $O(|\mathcal{M}|)$ time complexity.

B. HIERARCHICAL WEIGHT ALLOCATION

Whenever a new task is generated, the CTA algorithm selects a target device and assigns the task to the device. The execution of the newly assigned task may interfere with existing tasks in the device. Hence, when a new task arrives, a device/server needs to reschedule all the tasks including the new one, in order to maximize the number of deadline-met tasks. Rescheduling is initiated after all the input data required for a new task has been received from the communication module and, therefore, we do not need to consider the input data communication time while scheduling. Also, the communication time for output data can be regarded as a constant value since its size is known in advance via task profiling. Hence, for each task we can just subtract the communication time for output data from its deadline to make a refined deadline (D'_i) for scheduling (i.e., $D'_i = D_i - T^{i,out}_{A,\epsilon}$). Then it is enough to consider only the required CPU cycle (C_i) of each task to calculate a task schedule that maximizes the number of deadline-met tasks.

Our collaborative task scheduling assumes that, for generality, IoT devices run general-purpose operating systems that execute multiple tasks using time sharing-based scheduling algorithms. Then, rather than executing tasks one by one, it is more natural to make them run concurrently sharing computational resources. Task scheduling, in this case, can be enforced by differentiating CPU time allocated to each task, i.e., assigning different weights of CPU share to each task. Ignoring the CPU scheduling overhead, serial and concurrent task executions have the same total task execution time, and so the concurrent executions do not noticeably hurt the accuracy of remaining time estimation in the CTA algorithm that assumes the serial task execution.

The Hierarchical Weight Allocation (HWA) algorithm dynamically and hierarchically assigns weights to tasks considering their respective *refined* deadlines.

1) TASK CLASSIFICATION

Whenever a new task is assigned for execution, the Task Executor classifies the task into one of three task groups: *local task group* (LTG), *time critical task group* (TCG) and *best effort task group* (BEG). To guarantee the safe execution of local tasks in each IoT device, they are classified into LTG with the highest priority being assigned. Time critical tasks that have their own deadlines are classified into TCG, and other tasks are classified into BEG with the lowest priority assigned. Tasks in higher priority groups are guaranteed to use as much available CPU cycles as they need, and the remaining cycles are inherited to lower priority groups, hence forming a hierarchy among three task groups: LTG→TCG→BEG. It is noteworthy that a device certainly has enough CPU power to execute its local tasks that have the highest priority and, in our HWA algorithm, other tasks share only the remaining CPU power for their execution. Hence, the fair share of CPU cycles is enough among local tasks for their safe executions. Tasks in BEG are also enough to fairly share the inherited CPU cycles, if remain, from the TCG. Therefore, we only need to concentrate on how to share inherited CPU cycles from LTG among tasks in TCG to maximize the number of deadline-met tasks.

2) DYNAMIC WEIGHT ALLOCATION

The HWA algorithm maintains two logical task queues —*run queue* and *wait queue*—and all tasks in a device share the CPU cycles according to their respective share weight. Whenever task scheduling occurs, share weights are calculated and assigned to all the tasks such that tasks in the run queue have positive values whose sum is 1 and those in wait queue have zero. Let T_S be the system-defined time slice (in seconds) in which all tasks in the run queue are executed once, and W_i be the share weight of task t_i. Then, t_i runs on CPU for $W_i \times T_S$ of time during a single time slice.

The HWA algorithm pursues three design objectives; 1) guaranteeing the safe execution of *intermittent* local tasks,[1] 2) maximizing the number of deadline-met time critical tasks, and 3) exploiting as much CPU cycles as possible to maximize the overall task throughput. To achieve the first one, when there is any intermittent local task being executed, the HWA algorithm does not execute any offloaded tasks by moving them to the wait queue. Since tasks in the wait queue have zero weight, they are not executed until they move to the run queue through task scheduling in the future that occurs as soon as all local tasks finish. If there are multiple local tasks, the HWA algorithm assigns the same share weight to them so that they fairly share the CPU cycles.

When there is no intermittent local task, the HWA algorithm calculates the share weight of each task in TCG. For a task t_i in TCG, the HWA algorithm first calculates its minimum amount of CPU cycles required in each time slice, denoted by C_i^{min}, for a task to meet its refined deadline (D'_i). Since the remaining time until the refined deadline of t_i can be represented in terms of the number of time slices as $(D'_i - T_{cur})/T_S$, C_i^{min} becomes

$$C_i^{min} = \frac{C_i^R}{(D'_i - T_{cur})/T_S} = \frac{T_S \cdot C_i^R}{D'_i - T_{cur}},$$

where C_i^R denotes the remaining required CPU cycles of t_i until it finishes, which we can get by subtracting the total amount of CPU cycles spent by t_i from C_i. If we cannot allocate C_i^{min} CPU cycles to task t_i in every time slice, the task cannot avoid missing its deadline. Hence to make all tasks in TCG meet their deadlines, the sum of their minimum CPU cycle requirements must not exceed the amount of available CPU cycles in a time slice:

$$\sum_{t_i \in \mathcal{T}_A^{TCG}} C_i^{min} \leq (f_A - C_A^{avg}) \cdot T_S, \tag{12}$$

where $\mathcal{T}_A^{TCG}$ denotes the set of all tasks in TCG and $f_A - C_A^{avg}$ is the available CPU cycles in a second. If the constraint (12)

[1]The protection of *ever-running* local tasks is done by steadily reserving C_A^{avg} of CPU cycles in every time slice. See equation (12).

is not satisfied due to an excessive number of time critical tasks, the HWA algorithm selects one or more tasks and enqueues them to the wait queue. They can move to the run queue again, after a task finishes its execution leaving the CPU cycles occupied by the task available. For that reason, rescheduling tasks needs to occur also whenever an existing task finishes. To minimize the number of tasks that need to be enqueued to the wait queue, the HWA algorithm selects them one by one in their C_i^{min} order, until the constraint (12) is satisfied, and the remaining tasks are enqueued to the run queue.

When the constraint (12) is satisfied, the HWA algorithm adjusts the share weights of tasks in the run queue by normalizing their minimum CPU cycle requirements as:

$$W_i = \frac{C_i^{min}}{\sum_{t_i \in \mathcal{T}_A^{RQ}} C_i^{min}} \cdot \frac{f_A - C_A^{avg}}{f_A}$$

where $\mathcal{T}_A^{RQ}$ is the set of tasks in the run queue. Through the normalization process, a task t_i in the run queue is assigned $W_i T_S$ time in each time slice and can use up to $W_i T_S f_A$ cycles in a given time slice. Since $\sum_{t_i \in \mathcal{T}_A^{RQ}} C_i^{min} \leq (f_A - C_A^{avg})T_S$ by the constraint (12), $W_i T_S f_A \geq C_i^{min}$, which enables t_i to meet its refined deadline. And since the above normalization enables running tasks to exploit the entire available CPU cycles in each time slice (i.e., $(f_A - C_A^{avg})T_S$), the overall task execution time in the run queue is minimized.

The best-effort tasks in BEG are executed only when there is no tasks in both LTG and TCG. To this end, if there is any task in either LTG or TCG, best-effort tasks are enqueued to the wait queue. When the two task groups become empty, the HWA algorithm moves all tasks in BEG to the run queue assigning the same share weight to them.

V. PERFORMANCE EVALUATION

In this section, we present the performance of the collaborative task scheduling scheme through extensive experiments. Since, to the best of our knowledge, there is no prior work that assumes multiple tasks being executed concurrently in an IoT device, we compare our scheduling scheme (HWA) with three traditional scheduling policies, EDF (Earliest Deadline First), SJF (Shortest Job First) and FIFO. For task assignment, we either use our CTA algorithm or the weighted round-robin (WRR) algorithm which assigns tasks in proportion to the CPU power in a round-robin manner.

A. EXPERIMENTAL ENVIRONMENT

We implemented each component of the software architecture in Figure 2 on a single 32-core (2.1 GHz) Linux (Ubuntu 20.04.1 LTS) machine. The experimental testbed consists of four cloud servers, one edge server, and eight IoT devices, all of which run on 14 Linux Docker containers in the multicore server machine. Specifically, we create one container for each task executor in cloud servers and IoT devices. We also create two containers for an edge server, one for task manager and another for task executor. We assign two CPU cores to each container. When generating containers for IoT devices, we set their `cfs-quota_us` to one-third of the cloud/edge containers to represent their weak CPU power. Hence, the CPU frequencies of cloud server (f_c), edge server (f_ϵ) and IoT devices (f_d) are 2.1 GHz, 2.1 GHz and 0.7 GHz, respectively.

The edge server's task manager has three processes: task generator, task distributor, and communication module. The task generator generates edge tasks, each repeatedly calculating the greatest common divisor of two large numbers. We used the number of such repetitions to determine each task size. Specifically, we set the number of repetitions such that the required CPU cycles of offloaded tasks (C_i) follow Gaussian distribution with $\mu = 350$ Mcycles and $\sigma = 210$ Mcycles. And their deadlines are set to $C_i/f_d + (0.1 \sim 0.5)$ seconds, which are their execution times in an IoT device with random margin (0.1~0.5 sec.) added. For local tasks in the IoT device, we set their CPU cycle and the deadline to 500 Mcycles and 1 second, respectively. The task distributor determines where to assign each task among cloud, edge, and IoT devices based on either WRR or our CTA algorithm. A container for task executor also has a process for communication module and the task executor process. The communication module transfers tasks and outputs between the edge and cloud or IoT containers through Redis queues. The network latency between cloud and edge containers and between the edge and IoT containers is set to 128 *ms* and 3 *ms*, respectively, using Linux `tc` command.

Total 7,000 edge tasks are independently generated according to the poisson process with varying arrival rate from 1,000 to 4,000 tasks/min, and additional local tasks are randomly generated in each IoT device with 6 tasks/min arrival rate during the simulation. Among 7,000 edge tasks, 90% of tasks are time critical tasks and remaining 10% of tasks are best effort tasks. We repeated all experiments ten times and averaged their results.

B. PERFORMANCE COMPARISON

We evaluate the performance of our scheme in terms of both task throughput and deadline satisfaction ratio, by comparing eight cases of algorithm combination between four task scheduling and two task assignment algorithms.

1) THROUGHPUT

We first examine the throughput of the IoT-assisted edge computing. In this experiment, we measured the total time to complete all edge tasks (during which independently generated local tasks are also executed) and divided the number of all tasks by the measured time to get task throughput per second. Figure 5a shows the throughput of each case as task arrival rate (i.e., overall system load) increases. Aggregating the CPU frequencies of all nodes, we can estimate the theoretical maximum computation capacity of our experimental system, which is 2.1 Gcycles (edge) + 4 $\times$ 2.1 Gcycles(cloud) + 8 $\times$ 0.7 Gcycles (IoT) = 16.1 Gcycles in a second. Since the average required CPU cycles of offloaded tasks are 350 Mcycles, the hypothetical system's theoretical throughput bound

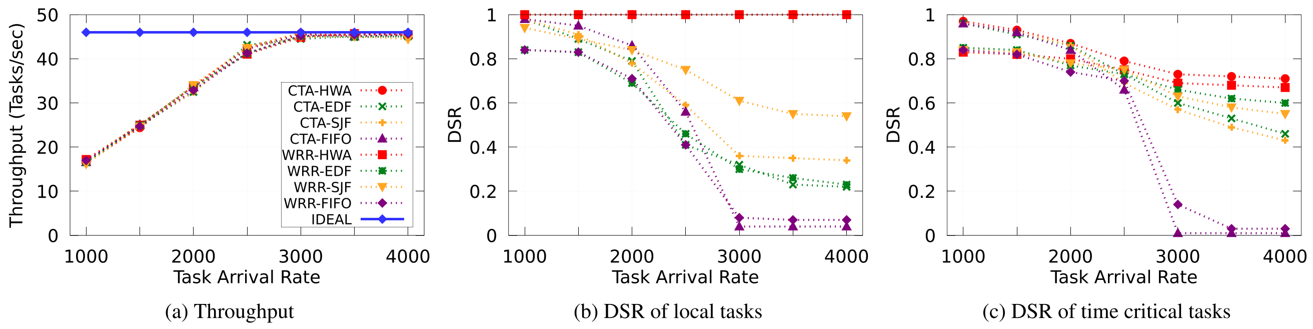


FIGURE 5. Comparison of the throughput and deadline satisfaction ratio with varying task arrival rates.

becomes about 46 tasks/sec, as plotted with 'IDEAL' title in Figure 5a.

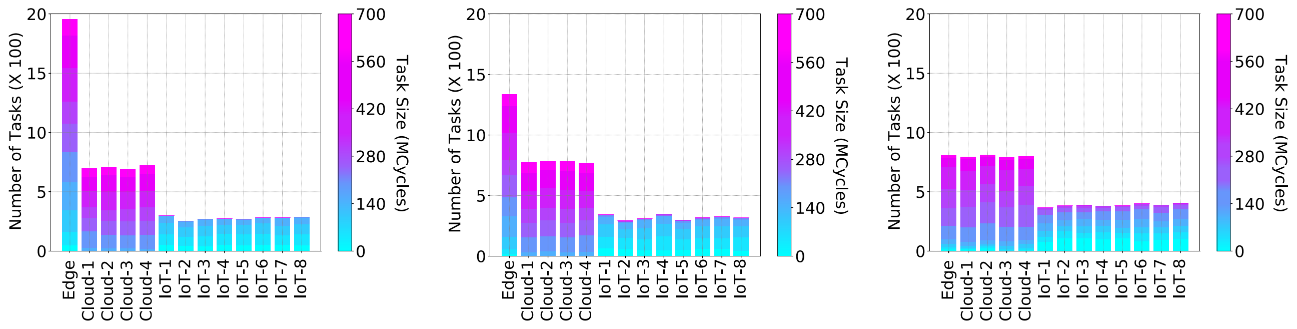
Since every task scheduling algorithm always fully exploits the CPU power as long as there exist tasks to be executed, scheduling algorithm itself does not affect the task throughput. Meanwhile, the task assignment algorithm may affect the task throughput since skewed task distribution can make lightly loaded devices idle and therefore increasing the total execution time. The WRR algorithm, which uniformly distributes tasks across nodes in proportion to their CPU power, naturally shows near-optimal throughput. Of importance is the fact that our CTA algorithm, which is designed to maximize the probability of each time critical task to meet its deadline, also shows near-optimal throughput. It means that the CTA algorithm does not sacrifice task throughput to achieve its design goal.

2) DEADLINE SATISFACTION RATIO (DSR)

Noticeable difference among scheduling algorithms is in their deadline satisfaction ratio, as shown in Figures 5b and 5c, which shows DSRs of local and time critical tasks, respectively. First of all, meeting deadlines of local tasks is of utmost importance so that IoT-assisted edge computing be feasible. As shown in Figure 5b, our HWA algorithm, either with CTA or with WRR, always meets their deadlines regardless of the system load, owing to its prioritization on the local tasks. However, other algorithms show rapid decrease in DSR as task arrival rate increases mainly due to their serial executions of tasks. Specifically, when an IoT device is executing an offloaded task a newly generated local task cannot be executed immediately but delayed until the offloaded task finishes, which can make the local task miss its deadline even under the lightly loaded situation. Since SJF does not consider deadlines, short running edge tasks, even arrived later than local tasks, may further delay executions of local tasks in SJF. Such delay of local task execution becomes more problematic as the sizes of local tasks get larger. Since EDF schedules tasks based on their deadlines, local tasks can be delayed and miss their deadlines by time critical tasks that have earlier deadlines than them. As system load increases the number of such time critical tasks also increases and accordingly, DSR of EDF algorithm rapidly decreases. Although SJF seems to outperform EDF in this experiment, the result totally changes when local tasks have larger sizes and shorter deadlines. Of importance here is the fact that HWA algorithm always guarantees deadlines of local tasks independent of such task characteristics.

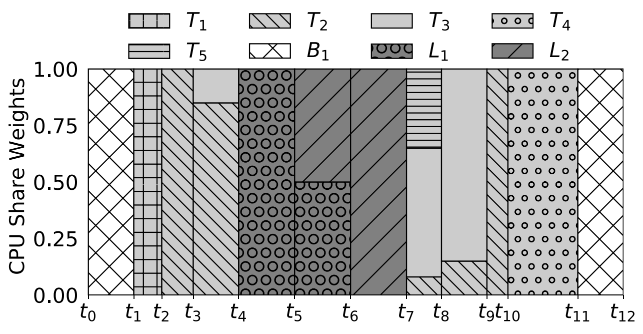
Figure 5c shows the DSR of time critical tasks. The combination of CTA and HWA algorithms outperforms all other combinations. Interesting here is the change of the dominant factor in determining DSR between task assignment and scheduling algorithms as system load increases. When the system is lightly loaded task assignment algorithms largely determine the DSR, whereas task scheduling algorithms differentiate DSR in heavily loaded situation. Specifically, all scheduling algorithms combined with our CTA algorithm show clearly better DSR than those with WRR algorithm until task arrival rate reaches 2000. In contrast, the HWA algorithm outperforms other scheduling algorithms, regardless of the task assignment algorithm, when the task arrival rate is higher than 2500. When the system is lightly loaded and therefore a small number of tasks are offloaded to each device, the deadline miss occurs mostly when multiple tasks with large execution time are offloaded to the same device. Since CTA algorithm assigns tasks based on their expected finish time in each device, tasks with large execution time tend to be distributed over all devices. However, using WRR algorithm that does not consider any task characteristics, such undesirable task assignment can occur and therefore show lower DSR than CTA algorithm regardless of the scheduling algorithm. Meanwhile, when the system is overloaded and so each device has a large number of tasks exceeding its capacity, their execution order (i.e., scheduling algorithm) mostly determines DSR.

The experimental result in Figure 5c delivers an important insight in designing IoT-assisted edge computing: *To maximize the deadline satisfaction ratio, not only task scheduling but also task assignment algorithms should be carefully designed incorporating characteristics of tasks to be offloaded.* CTA and HWA algorithms can be good candidates for such task assignment and scheduling algorithms, respectively.



(a) Task Arrival Rate = 1,000 (b) Task Arrival Rate = 1,500 (c) Task Arrival Rate = 2,500

FIGURE 6. **Number and sizes of tasks assigned to each device by CTA: Total 7,000 tasks.**



	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}
Arr.	B_1	T_1	T_2	T_3,T_4	L_1	L_2	T_5						
Fin.			T_1				L_1	L_2	T_5	T_3	T_2	T_4	B_1

FIGURE 7. **Dynamic weights assignment by HWA algorithm: T, B and L represent time critical, best effort and local tasks, respectively. *x*-axis represents times when events (task arrival or finish) occurred.**

C. BEHAVIOR OF CTSF

In this section, we present the detailed behavior of task allocation (CTA) and scheduling (HWA) algorithms.

1) TASK DISTRIBUTION BY CTA

The CTA algorithm assigns a new task to the device where the task can be finished earliest. Figure 6 shows the total number of tasks assigned to each device and their distribution of sizes in each device. The CTA algorithm prefers the edge server because the communication overhead for task offloading and output data transfer does not exist when a task is executed in the edge server. However, when the task load in the edge server is large, the CTA algorithm assigns tasks to other devices considering their completion times. Hence, as the task arrival rate increases, more and more tasks are assigned to either cloud servers or IoT devices.

Interesting is the size distribution of tasks in each device. Note that cloud servers have stronger CPU power but larger communication latency than IoT devices. A small-sized task has larger finish time when it is assigned to a cloud server due to its large communication latency. However, a large task can benefit from strong CPU power of the cloud server compensating communication latency. Therefore, small tasks are likely to be assigned to IoT devices whereas large tasks are assigned to cloud servers, which cannot be observed when using WRR algorithm for task assignment. Such task assignment behavior of CTA algorithm increases the DSR of time critical tasks as shown in Figure 5c.

2) TASK SCHEDULING BY HWA

Figure 7 shows dynamically changing share weights of 8 tasks (five time critical, one best effort and two local tasks), arrived in their number order, by HWA algorithm. Whenever a new task arrives or existing task finishes, the HWA algorithm calculates and assigns new weights to tasks. It behaves as follows: 1) local tasks (L_1 and L_2), as soon as arrived, monopolize the entire CPU cycles with fair share until they finish, pausing (T_2 and T_3) or delaying (T_5) all other tasks by assigning zero weights to them, 2) time critical tasks share CPU cycles based on their respective required CPU cycles and deadline and 3) executions of best effort tasks are paused (B_1) or postponed (i.e., assigned zero weights) until there remains no time critical or local task.

Noticeable here is the out of order executions of T_4 and T_5. At the time when T_4 arrived (t_3), the sum of minimum CPU cycle requirements (C_i^{min}) of all time critical tasks (T_2, T_3 and T_4) exceeds the amount of CPU cycles in a time slice (i.e., (12) is not satisfied), and HWA algorithm moves T_4, which has the largest C_i^{min} value, to the wait queue and only T_2 and T_3 are executed. After that when N_2 finishes (t_7), there are four time critical tasks (T_2, T_3, T_4 and T_5). The HWA algorithm finds that, while it cannot meet deadlines of all of them, T_2, T_3 and T_5 can meet their deadlines. Therefore it leaves T_4, which has the largest C_i^{min} value, in the wait queue. When T_3 finishes at time t_9, since we cannot meet the deadlines of all remaining time critical tasks, T_2 and T_4, the HWA algorithm still leaves T_4 in the wait queue until T_2 finishes, and then moves T_4 to the run queue and executes it at t_{10}. The paused best effort task B_1 is resumed when T_4 finishes at t_{11}.

The HWA algorithm not only guarantees deadline-met executions of local tasks as shown in Figure 5b but also achieves better deadline satisfaction ratio for time critical tasks than other scheduling policies (Figure 5c) owing to its dynamic weight allocation.

D. EFFECT TO THE EDGE COMPUTING PERFORMANCE

For IoT-assisted edge computing be feasible, two conditions should be satisfied: 1) offloaded tasks to IoT devices do not hurt normal execution of local tasks in each IoT device, i.e., local tasks in an IoT device should be able to meet their respective deadlines despite executions of offloaded tasks, and 2) computing resources in IoT devices should be effectively exploited for executing offloaded tasks for edge services, and therefore increase the throughput of edge computing services. Figure 5 confirms that the first condition is satisfied when using our HWA algorithm. To examine the second condition, we measured the task throughput varying the number of participating IoT devices from zero to eight. When there is no IoT device all tasks are executed in either edge server or cloud servers. Figure 8 shows the results. As more IoT devices are used, the task throughput monotonically increases, satisfying the second condition. Results in Figures 5 and 8 confirm that IoT-assisted edge computing with our collaborative task scheduling scheme being used is feasible.

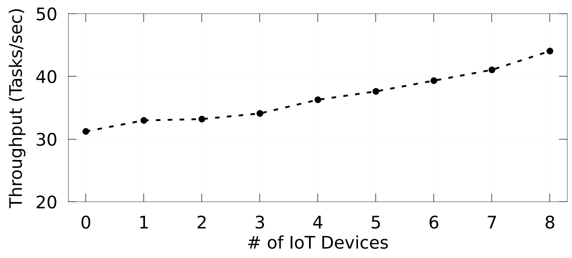


FIGURE 8. Throughput of IoT-assisted edge computing: Task arrival rate = 4,000 tasks per minute.

VI. CONCLUSION

As more and more IoT devices with abundant computational resources are emerged, the effective exploitation of unused resources in IoT devices for other meaningful work than the local IoT job is of increasing importance. In this work, we propose to use those unused resources for empowering edge computing, i.e., IoT-assisted edge computing, by offloading a portion of edge tasks to IoT devices. To that end, we propose a collaborative task scheduling scheme in which edge server assigns tasks using completion time-based task assignment algorithm and IoT devices take the role of task scheduling using hierarchical weight allocation algorithm. Experimental results show that the proposed scheme achieves increased edge service throughput by exploiting unused resources in IoT devices while guaranteeing deadlines of local IoT tasks.

REFERENCES

[1] H. Li, K. Ota, and M. Dong, "Learning IoT in edge: Deep learning for the Internet of Things with edge computing," *IEEE Netw.*, vol. 32, no. 1, pp. 96–101, Jan. 2018.

[2] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 1, pp. 856–868, Jan. 2019.

[3] X. Lyu, H. Tian, C. Sengul, and P. Zhang, "Multiuser joint task offloading and resource optimization in proximate clouds," *IEEE Trans. Veh. Technol.*, vol. 66, no. 4, pp. 3435–3447, Apr. 2017.

[4] N. Auluck, A. Azim, and K. Fizza, "Improving the schedulability of real-time tasks using fog computing," *IEEE Trans. Services Comput.*, early access, Sep. 27, 2019, doi: 10.1109/TSC.2019.2944360.

[5] M. Grob. (2017). *We Are Making on-Device Ai Ubiquitous*. Accessed: Jul. 23, 2019. [Online]. Available: https://www.qualcomm.com/news/onq/2017/08/16/we-are-making-device-ai-ubiquitous

[6] A. Kendall and Y. Gal, "What uncertainties do we need in Bayesian deep learning for computer vision?" in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5574–5584.

[7] G. A. McGilvary, A. Barker, and M. Atkinson, "Ad hoc cloud computing," in *Proc. IEEE 8th Int. Conf. Cloud Comput.*, Jun. 2015, pp. 1063–1068.

[8] M. Mehrabi, D. You, V. Latzko, H. Salah, M. Reisslein, and F. H. P. Fitzek, "Device-enhanced MEC: Multi-access edge computing (MEC) aided by end device computation and caching: A survey," *IEEE Access*, vol. 7, pp. 166079–166108, 2019.

[9] P. Menage, P. Jackson, and C. Lameter. (2008). *Cgroups*. [Online]. Available: http://www.mjmwired.net/kernel/Documentation/cgroups.txt

[10] S. Guo, B. Xiao, Y. Yang, and Y. Yang, "Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing," in *Proc. IEEE INFOCOM*, Apr. 2016, pp. 1–9.

[11] Y. Yu, J. Zhang, and K. B. Letaief, "Joint subcarrier and CPU time allocation for mobile edge computing," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2016, pp. 1–6.

[12] L. Yang, H. Zhang, M. Li, J. Guo, and H. Ji, "Mobile edge computing empowered energy efficient task offloading in 5G," *IEEE Trans. Veh. Technol.*, vol. 67, no. 7, pp. 6398–6409, Jul. 2018.

[13] L. Ji and S. Guo, "Energy-efficient cooperative resource allocation in wireless powered mobile edge computing," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4744–4754, Jun. 2019.

[14] H. Guo, J. Zhang, J. Liu, and H. Zhang, "Energy-aware computation offloading and transmit power allocation in ultradense IoT networks," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4317–4329, Jun. 2019.

[15] W. Chen, D. Wang, and K. Li, "Multi-user multi-task computation offloading in green mobile edge cloud computing," *IEEE Trans. Services Comput.*, vol. 12, no. 5, pp. 726–738, Sep. 2019.

[16] M. Molina, O. Munoz, A. Pascual-Iserte, and J. Vidal, "Joint scheduling of communication and computation resources in multiuser wireless application offloading," in *Proc. IEEE PIMRC*, Sep. 2014, pp. 1093–1098.

[17] Y. Sahni, J. Cao, and L. Yang, "Data-aware task allocation for achieving low latency in collaborative edge computing," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 3512–3524, Apr. 2019.

[18] Z. Ning, P. Dong, X. Kong, and F. Xia, "A cooperative partial computation offloading scheme for mobile edge computing enabled Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4804–4814, Jun. 2019.

[19] Q. Ju, G. Sun, H. Li, and Y. Zhang, "Collaborative in-network processing for Internet of battery-less things," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 5184–5195, Jun. 2019.

[20] Y. Zhang, X. Lan, Y. Li, L. Cai, and J. Pan, "Efficient computation resource management in mobile edge-cloud computing," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 3455–3466, Apr. 2019.

[21] 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.

[22] T. Quang Dinh, J. Tang, Q. Duy La, and T. Q. S. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Trans. Commun.*, vol. 65, no. 8, pp. 3571–3584, Aug. 2017.

[23] X. Wei, C. Tang, J. Fan, and S. Subramaniam, "Joint optimization of energy consumption and delay in cloud-to-thing continuum," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2325–2337, Apr. 2019.

[24] Y. Gao, W. Tang, M. Wu, P. Yang, and L. Dan, "Dynamic social-aware computation offloading for low-latency communications in IoT," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 7864–7877, Oct. 2019.

[25] D. Zhang, Y. Ma, C. Zheng, Y. Zhang, X. S. Hu, and D. Wang, "Cooperative-competitive task allocation in edge computing for delay-sensitive social sensing," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Oct. 2018, pp. 243–259.

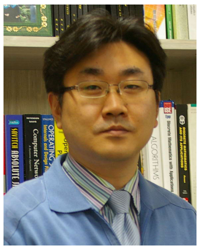
[26] D. Zhang, Y. Ma, Y. Zhang, S. Lin, X. S. Hu, and D. Wang, "A real-time and non-cooperative task allocation framework for social sensing applications in edge computing systems," in *Proc. IEEE RTAS*, Apr. 2018, pp. 316–326.
[27] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proc. ACM Eurosys*, 2011, pp. 301–314.
[28] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervas. Comput.*, vol. 8, no. 4, pp. 14–23, Oct. 2009.
[29] D. Huang, P. Wang, and D. Niyato, "A dynamic offloading algorithm for mobile computing," *IEEE Trans. Wireless Commun.*, vol. 11, no. 6, pp. 1991–1995, Jun. 2012.
[30] W. Zhang, Y. Wen, and D. O. Wu, "Collaborative task execution in mobile cloud computing under a stochastic wireless channel," *IEEE Trans. Wireless Commun.*, vol. 14, no. 1, pp. 81–93, Jan. 2015.
[31] W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, and D. O. Wu, "Energy-optimal mobile cloud computing under stochastic wireless channel," *IEEE Trans. Wireless Commun.*, vol. 12, no. 9, pp. 4569–4581, Sep. 2013.
[32] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proc. USENIX NSDI*, 2005, pp. 273–286.
[33] E. Cuervo, A. Balasubramanian, D.-K. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making smartphones last longer with code offload," in *Proc. 8th Int. Conf. Mobile Syst., Appl., Services*, 2010, pp. 49–62.
[34] N. Wang, B. Varghese, M. Matthaiou, and D. S. Nikolopoulos, "ENORM: A framework for edge node resource management," *IEEE Trans. Services Comput.*, early access, Sep. 18, 2017, doi: 10.1109/TSC.2017.2753775.
[35] M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta, and A. Neal, "Mobile-edge computing introductory technical white paper," Mobile-Edge Comput. Ind. Initiative, ETSI, Sophia Antipolis, France, White Paper 1089–7801, 2014.
[36] H.-S. Lee and J.-W. Lee, "Resource and task scheduling for SWIPT IoT systems with renewable energy sources," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2729–2748, Apr. 2019.
[37] G. Fortino, C. Savaglio, C. E. Palau, J. S. de Puga, M. Ganzha, M. Paprzycki, M. Montesinos, A. Liotta, and M. Llop, "Towards multi-layer interoperability of heterogeneous IoT platforms: The inter-IoT approach," in *Integration, Interconnection, and Interoperability of IoT Systems*. Berlin, Germany: Springer, 2018, pp. 199–232.
[38] M. B. Alaya, S. Medjiah, T. Monteil, and K. Drira, "Toward semantic interoperability in onem2m architecture," *IEEE Commun. Mag.*, vol. 53, no. 12, pp. 35–41, Dec. 2015.
[39] S. Pallewatta, V. Kostakos, and R. Buyya, "Microservices-based iot application placement within heterogeneous and resource constrained fog computing environments," in *Proc. IEEE/ACM UCC*, Dec. 2019, pp. 71–81.
[40] M. Xiao, N. B. Shroff, and E. K. P. Chong, "A utility-based power-control scheme in wireless cellular systems," *IEEE/ACM Trans. Netw.*, vol. 11, no. 2, pp. 210–221, Apr. 2003.
[41] Y. Mao, J. Zhang, S. H. Song, and K. B. Letaief, "Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems," *IEEE Trans. Wireless Commun.*, vol. 16, no. 9, pp. 5994–6009, Sep. 2017.

YOUNGJIN KIM received the B.S. and M.S. degrees in computer science from Hanyang University, Seoul, South Korea, in 2013 and 2015, respectively, where he is currently pursuing the Ph.D. degree in computer science. His research interests include distributed computing systems, storage systems for cloud computing, big data processing, edge computing, and cloud computing.

CHIWON SONG received the B.S. degree in industrial engineering and the M.S. degree in computer science from Hanyang University, Seoul, South Korea, in 2010 and 2012, respectively, where he is currently pursuing the Ph.D. degree. He is currently working for LG Electronics, South Korea. His research interests include cloud computing, security and privacy for cloud services, and distributed computing systems.

HYUCK HAN received the B.S., M.S., and Ph.D. degrees in computer science and engineering from Seoul National University, Seoul, South Korea, in 2003, 2006, and 2011, respectively. From 2011 to 2013, he worked for Samsung as a Senior Researcher. Since March 2014, he joined the Department of Computer Science, Dongduk Women's University, Seoul, South Korea, as an Assistant Professor. His research interests are operating systems, database systems, and distributed systems.

HYUNGSOO JUNG received the B.S. degree in mechanical engineering from Korea University, Seoul, in 2002, and the M.S. and Ph.D. degrees in computer science from Seoul National University, South Korea, in 2004 and 2009, respectively. From 2010 to 2012, he was with The University of Sydney, Sydney, Australia, as a Postdoctoral Research Associate. From April 2012 to September 2012, he was a Researcher at NICTA. From October 2012 to August 2015, he worked for Amazon Web Services as a (Senior) Software Development Engineer. Since September 2015, he joined Hanyang University as an Assistant Professor. His research interests are in the areas of distributed systems, database systems, and transaction processing.


SOOYONG KANG (Member, IEEE) received the B.S. degree in mathematics and the M.S. and Ph.D. degrees in computer science from Seoul National University (SNU), Seoul, South Korea, in 1996, 1998, and 2002, respectively. He was then a Postdoctoral Researcher with the School of Computer Science and Engineering, SNU. Since March 2003, he joined the Department of Computer Science, Hanyang University, Seoul, as a Professor. From January 2017 to February 2018, he was a Visiting Professor with the Department of Computing, Macquarie University, Sydney, Australia. His research interests include storage systems, distributed computing systems, mobile cloud computing, edge computing, and the Internet of Things.