Profit-Maximized Collaborative Computation Offloading and Resource Allocation in Distributed Cloud and Edge Computing Systems

Haitao Yuan, *Member, IEEE*, and MengChu Zhou, *Fellow, IEEE*

***Abstract*— Edge computing is a new architecture to provide computing, storage, and networking resources for achieving the Internet of Things. It brings computation to the network edge in close proximity to users. However, nodes in the edge have limited energy and resources. Completely running tasks in the edge may cause poor performance. Cloud data centers (CDCs) have rich resources for executing tasks, but they are located in places far away from users. CDCs lead to long transmission delays and large financial costs for utilizing resources. Therefore, it is essential to smartly offload users' tasks between a CDC layer and an edge computing layer. This work proposes a cloud and edge computing system, which has a terminal layer, edge computing layer, and CDC layer. Based on it, this work designs a profit-maximized collaborative computation offloading and resource allocation algorithm to maximize the profit of systems and guarantee that response time limits of tasks are strictly met. In each time slot, this work jointly considers CPU, memory, and bandwidth resources, load balance of all heterogeneous nodes in the edge layer, maximum amount of energy, maximum number of servers, and task queue stability in the CDC layer. Considering the abovementioned factors, a single-objective constrained optimization problem is formulated and solved by a proposed simulated-annealing-based migrating birds optimization procedure to obtain a close-to-optimal solution. The proposed method achieves joint optimization of computation offloading between CDC and edge, and resource allocation in CDC. Realistic data-based simulation results demonstrate that it realizes higher profit than its peers.**

***Note to Practitioners*—This work considers the joint optimization of computation offloading between Cloud data center (CDC) and edge computing layers, and resource allocation in CDC. It is important to maximize the profit of distributed cloud and edge computing systems by optimally scheduling all tasks between them given user-specific response time limits of tasks. It is challenging to execute them in nodes in the edge computing layer because their computation resources and battery capacities are often constrained and heterogeneous. Current offloading methods fail to jointly optimize computation offloading and resource allocation for nodes in the edge and servers in CDC. They are insufficient and coarse-grained to schedule arriving tasks. In this work, a novel algorithm is proposed to maximize the profit of distributed cloud and edge computing systems while meeting response time limits of tasks. It explicitly specifies the task service rate and the selected node for each task in each time slot by considering resource limits, load balance requirement, and processing capacities of nodes in the edge, and server and energy constraints in CDC. Real-life data-driven simulations show that the proposed method realizes a larger profit than several typical offloading strategies. It can be readily implemented and incorporated into large-scale industrial computing systems.**

***Index Terms*— Cloud data centers (CDCs), computation offloading, edge computing, intelligent optimization, migrating birds optimization (MBO), simulated annealing (SA).**

I. Introduction

With the fast development of information and communication technologies, smart mobile devices (SMDs) have been gaining enormous attention with current mobile technologies, e.g., wearable devices and Internet of Things (IoT) [1] as they enable convenient communications almost anywhere and anytime. These technologies give a powerful platform for smart mobile applications [2] (e.g., speech/face recognition, augmented/assisted/virtual reality, social networking, traffic management, path planning, and health monitoring). The gap between limited computing resources and need for running complex applications is increasing [3]. It is challenging to execute them in SMDs that own limited computation resources and battery capacities. It is shown that cloud data centers (CDCs), e.g., Amazon EC2 and Microsoft Azure, provide new distributed computing to efficiently tackle the limitation of battery and processing capabilities by offloading part or all computation-intensive tasks to CDCs for execution [4]. In 2019, over 70% of calculations have been completed in CDCs. The applications send their tasks to remote CDCs that typically have rich computation resources, high security, and huge storage [5]. However, CDCs are usually far away from SMDs, and this leads to unacceptable transmission delay and economic cost for utilizing resources in CDCs [6] and affects the real-time performance of latency-sensitive applications.

To tackle the shortcomings of CDCs, edge computing is an emerging architecture for the network edge, and it provides

Manuscript received May 7, 2020; accepted June 4, 2020. Date of publication July 14, 2020; date of current version July 2, 2021. This article was recommended for publication by Associate Editor X. Li and Editor Q. Zhao upon evaluation of the reviewers' comments. *(Corresponding author: Haitao Yuan.)*

The authors are with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102 USA (e-mail: haitao.yuan@njit.edu; zhou@njit.edu).

Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASE.2020.3000946

agile and pervasive resources to IoT applications with strict latency need at anytime and anywhere [7]. The need for real-time and scalable data analysis in IoT devices is a major driving power for edge computing where data is generated and processed in the network edge in many applications of smart city, smart home, surveillance networks, connected vehicles, and industrial IoT. Compared with powerful computing platforms in CDCs, edge computing enables higher computing agility and lower latency. It is shown that about 40% of IoT-produced data are stored and processed at the edge of a network [8]. Local computing in the edge greatly reduces the response time because waiting or communication delay between the edge and CDCs is avoided. Yet, the limits of energy, computation, and storage resources of nodes in IoT devices restrict the number of tasks of resource-hungry applications locally computed in the edge. Therefore, it is unlikely to execute all tasks in nodes in edge computing (i.e., local computing), and some of them have to be offloaded to CDCs to avoid energy depletion and the overall system performance [9]. Thus, it is critically important to rationally schedule all tasks between CDC and edge computing layers and maximize the profit of distributed cloud and edge computing systems while ensuring that user-specific response time limits of tasks are well met.

To solve this issue, this work focuses on a three-layer architecture. It consists of terminal, edge computing, and CDC layers. This work aims to maximize the profit of the system provider by smartly scheduling arriving tasks between CDC and edge computing layers while strictly guaranteeing their response time limits. More specifically, given tasks scheduled to execute in a CDC layer, this work explicitly specifies the task service rate of a CDC server in each time slot. In addition, given tasks scheduled to an edge computing layer, this work explicitly specifies the selected node for each task in each time slot. In this way, this work proposes a more fine-grained mechanism to obtain the optimal scheduling strategy for arriving tasks. Specifically, it jointly considers CPU, memory, and bandwidth resource limits, load balance requirements of all nodes, and different processing capacities of heterogeneous nodes in the edge computing layer. In addition, it jointly considers the maximum amount of energy, the maximum number of available servers, and the task queue stability of servers in the CDC layer. By jointly considering the abovementioned factors, this work proposes a profit-maximized collaborative computation offloading and resource allocation algorithm. The system profit is maximized, while the response time limits of tasks are strictly met. Real-life data, e.g., arriving tasks in Google cluster and prices of power grid, are adopted to evaluate it. The experimental results prove that it achieves higher profit than its typical scheduling peers can realize.

The rest of this work is organized as follows. The related studies are compared and discussed in Section II. Section III first gives the illustrative framework of the system and then formulates a constrained profit maximization problem. Section IV presents the details of Simulated-annealing-based Migrating Birds Optimization (SMBO) to efficiently solve it. Real-life data are adopted to evaluate it in Section V. Finally, the conclusion is drawn in Section VI.

II. Related Work

A. Computation Offloading

The computation offloading is of great importance in edge computing and has been attracting a growing amount of attention in recent years [9]–[12]. Zhao *et al.* [9] proposed a collaborative computation offloading method that offloads application services to automobiles in vehicular networks and optimizes resource allocation for mobile edge computing (MEC) and cloud computing. A distributed algorithm for computation offloading and resource allocation is designed to obtain the optimal solution. The computation time and system utility for computation-intensive tasks are improved effectively. Liu and Liu [10] designed a price-based distributed approach to schedule users' offloaded computation tasks. They formulate a Stackelberg game to analyze the interaction between the users and the edge cloud. The prices are set by the edge cloud to achieve revenue maximization by considering its finite computation capacity. Then, each user separately makes its own offloading decision to realize the minimization of latency and payment. Based on the network information of the edge cloud, differentiated and uniform pricing algorithms are developed and implemented in a distributed way. Wei *et al.* [11] modeled computation offloading as a Markov decision process (MDP), and reinforcement learning algorithms are used to obtain the optimal offloading decision. A polynomial value function approximation approach is developed to accelerate the learning process. Then, an after-state reinforcement learning mechanism for MDP is designed to obtain the optimal offloading strategy for real MEC systems. Zheng *et al.* [12] considered a multiuser computation offloading problem for mobile cloud computing in a dynamic environment. Mobile users are inactive or active dynamically, and their wireless channels to offload computation change randomly. An offloading decision process is formulated as a stochastic game for mobile users in a dynamic environment because each user selfishly makes its computation offloading decision. A multiagent stochastic algorithm is proposed to achieve the Nash equilibrium with an analytically derived convergence rate. Although recent studies consider the computation offloading for edge computing, they fail to achieve the joint optimization of computation offloading and resource allocation in CDC.

Our work jointly optimizes the computation offloading between CDC and edge computing layers, and resource allocation in the edge computing layer for latency-sensitive tasks. The resources include CPU, memory, and bandwidth at the edge computing layer and servers in the CDC layer.

B. Performance Optimization in Edge Computing

In edge computing, performance optimization is an important yet challenging topic, which involves evaluating performance and deciding where to execute users' tasks and how to allocate computing resources [13]–[16]. Responsiveness of tasks is important because applications are usually real time.

Tao *et al.* [13] analyzed the energy efficiency and performance guarantee of MEC. An energy minimizing optimization problem is formulated and solved by using the Karush–Kuhn–Tucker conditions for better performance of tasks and lower energy consumption. A request offloading scheme is designed by specifying bandwidth capacity and energy consumption at each time slot. Zhu *et al.* [14] designed a solution for quality and latency-optimized task scheduling in vehicular fog computing. The task allocation across mobile and stationary fog nodes are formulated as a biobjective optimization problem by considering constraints on quality loss, service latency, and fog capacity. An event-triggered dynamic task allocation method is proposed by applying binary particle swarm optimization (PSO) and linear programming-based optimization to achieve a tradeoff between quality loss and service latency. Ren *et al.* [15] formulated a joint computation and communication resource allocation problem to minimize the average latency of all mobile devices. A closed-form optimal task allocation policy is derived in terms of normalized cloud computation and backhaul communication capacities. In addition, the original problem is further transformed into an equivalent convex optimization one and solved by the convex optimization technique to obtain the optimal computation resource allocation. Chen *et al.* [16] modeled the optimal computation offloading as an MDP and aimed to maximize the long-term utility performance. The offloading decision is made according to energy and task queue state, and channel qualities between base stations and mobile users. Then, to avoid the high dimensionality curse in state space, a deep Q-network-based strategic algorithm is proposed to achieve computation offloading without the priori information of network dynamics.

Comparing with existing studies, this work aims to improve the system performance by proposing a smart offloading algorithm. It considers two offloading destinations (nodes in edge and servers in CDC) and provides a fine-grained model for response time of tasks executed in edge and CDC by jointly splitting tasks among nodes in edge and specifying the task service rate of servers in CDC.

C. Energy Optimization in Edge Computing

Optimizing energy consumption is one of the most challenging problems in edge computing because its nodes are typically equipped with limited battery energy [17]–[20]. Xu *et al.* [17] investigated a multiuser MEC system with the constraint of task latency. Users can reduce energy consumption by partially or completely offloading their tasks to an MEC server for edge computing while meeting the latency constraint. They formulate the joint optimization of data compression, computation offloading, and resource allocation as a problem with the constraints of latency and MEC computation capacity. This problem is then transformed into a convex one solved by convex optimization. Li *et al.* [18] incorporated MEC into virtualized cellular networks with machine-to-machine communications, to optimize computing resource allocation and reduce energy consumption. They formulate a random access process as partially observable MDP to minimize energy consumption and execution time of the system. Zhang *et al.* [19] considered a power consumption problem in a multiuser MEC system with energy harvesting devices. A problem of power consumption minimization with constraints of quality of service (QoS) and battery stability is formulated as a stochastic optimization program. An online algorithm based on the Lyapunov optimization is designed to solve the problem. It only needs current states of users. A distributed algorithm is proposed by using the alternating direction approach of multipliers to decrease the computational complexity. Zhou *et al.* [20] investigated the problem of energy-efficient workload offloading in vehicular networks. To solve it, a low-complexity distributed method is designed by using a consensus alternating direction approach of multipliers. Several local variables are incorporated for each user equipment, and the original problem is converted into an equivalent consensus problem with multiple objectives and different constraints. Then, the consensus problem is decomposed into many subproblems, which are solved simultaneously.

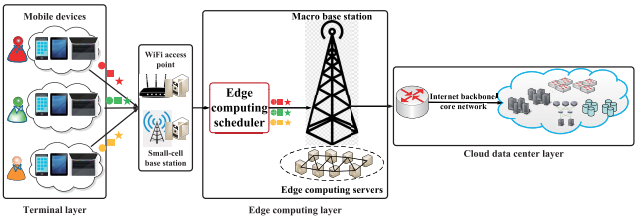

Fig. 1. Illustrative system framework.

Different from these studies, this work provides a higher accuracy and fine-grained energy model. It jointly considers CPU, memory, and bandwidth resource limits, load balance requirements of all nodes, and different processing capacities of heterogeneous nodes in the edge computing layer. In addition, it jointly considers the key parameters of CPU utilization, task service rate of CDC servers, task arriving rate, power usage effectiveness, average peak and idle power of each server, price of power grid, maximum amount of energy, maximum number of available servers, and task queue stability in the CDC layer.

III. Problem Formulation

This section gives the formulation of a constrained optimization problem for a cloud and edge computing system, as shown in Fig. 1. In this work, the framework includes three layers, i.e., terminal, edge computing, and CDC ones. The arriving tasks of users are sent through heterogeneous SMDs, e.g., iPad, smartphones, computers, sensors on the road, and all they are produced at the terminal layer, and the returned results are sent back to this layer eventually. These tasks need to be executed by such applications as industrial Internet devices, smart home, smart city, autonomous driving, and medical monitoring.

This work considers a heterogeneous network to deliver arriving tasks to nodes in edge computing and CDC layers. As shown in Fig. 1, for a given area, the wireless infrastructure of this network mainly includes many Wi-Fi access points and multiple small-cell base stations (SBSs). The cell radius of SBSs that might be hosted by different mobile telecom carriers varies from 0.01 to 2 km, and SBSs are mutually

interconnected and reachable to transmit signals and messages. SBSs, e.g., Wi-Fi access points, gateways, and microcells, are part of the network for highly populated urban areas. Tasks are delivered to the edge computing layer, including a task scheduler and nodes in edge computing. The scheduler smartly allocates tasks between the edge and CDC. If a task is scheduled to CDC, a macro base station (MBS) just forwards it to CDC; otherwise, an MBS needs to execute it in its node (server) at the edge computing layer.

MBS is located at the network edge and consists of many heterogeneous nodes with limited storage, computing, and transmission ability. It provides ubiquitous coverage and its cell radius varies from 8 to 30 km. It receives tasks from the terminal layer and executes some of them locally and sends others to CDC through the Internet backbone. It reduces tasks' latency and alleviates pressure on the CDC layer. The CDC layer owns multiple interconnected server clusters with large computing and storage capacities and provides different kinds of cloud resources to handle complex tasks.

A. Decision Variables

Let $o_\tau^{i,j}$ be a binary variable. If task i is scheduled to execute in node j in the edge computing layer in each time slot τ, $o_\tau^{i,j} = 1$; otherwise, $o_\tau^{i,j} = 0$. Let I_τ denote the number of tasks scheduled to nodes in the edge computing layer in time slot τ. Let μ_τ denote the task service rate of CDC servers in time slot τ. It is assumed that users' arriving tasks can be executed in the edge computing or CDC layer. Thus, in time slot τ, I_τ needs to be less than or equal to the total number of arriving tasks in τ and $\lambda_\tau \gamma$, that is

$$I_\tau \le \lambda_\tau \gamma \tag{1}$$

where λ_τ is the task service rate of CDC in each time slot τ and γ is the length of each time slot.

Let n_τ^j denote the maximum number of tasks that can be executed by node j in edge in time slot τ. Thus

$$\sum_{i=1}^{I_\tau} o_\tau^{i,j} \le n_\tau^j. \tag{2}$$

In time slot τ, task i has to be and can only be scheduled to execute in only one node in the edge computing layer. Thus

$$\sum_{j=1}^{J} o_\tau^{i,j} = 1. \tag{3}$$

B. Response Time

The total number of arriving tasks in time slot τ is $\lambda_\tau \gamma$ according to (1). Then, the number of tasks scheduled to execute in CDC in time slot τ is $\lambda_\tau \gamma - I_\tau$. Let λ_τ^c denote the arriving rate of tasks scheduled to CDC in time slot τ, i.e., $\lambda_\tau^c = (\lambda_\tau \gamma - I_\tau/\gamma)$. Let $\hat{T}_\tau$ denote the maximum response time of tasks executed in all nodes in the edge computing layer in time slot τ and $\widetilde{T}_\tau$ denote the average response time of all tasks executed in the CDC layer in time slot τ.

In the edge computing layer, there are J heterogeneous nodes. Let $T_\tau^{i,j}$ denote the execution time of task i $(1 \le i \le I_\tau)$ on node j $(1 \le j \le J)$, k_i denote the size of task i, and p_j denote the processing speed of node j. Then

$$T_\tau^{i,j} = \frac{k_i}{p_j} o_\tau^{i,j}. \tag{4}$$

It is worth noting that the maximum response time ($\hat{T}_\tau$) of all tasks executed in the edge computing layer includes transmission time and computation time in the CDC layer. Then, $\hat{T}_\tau$ is obtained as

$$\hat{T}_\tau = \max_{j=1}^{J} \left[\sum_{i=1}^{I_\tau} T_\tau^{i,j} \right]. \tag{5}$$

Let T^+ denote users' response time limit for tasks scheduled to execute in the edge computing layer. Thus, $\hat{T}_\tau$ cannot exceed T^+, that is

$$\hat{T}_\tau \le T^+. \tag{6}$$

Similar to [21] and [22], this work adopts an $M/M/1/\beta/\infty$ queuing system to analyze the behavior of switched-ON servers in CDC. Here, β denotes the maximum number of tasks that all servers in a CDC can execute. It is assumed that users' tasks arrive in a Poisson process with mean rate λ_τ^c and task service time has an exponential distribution with mean rate μ_τ. Let $\widetilde{T}_\tau$ denote the average response time of tasks scheduled to execute in the CDC layer and T^c denote users' response time limit of tasks scheduled to execute in the CDC layer. Then, $\widetilde{T}_\tau$ is obtained as

$$\widetilde{T}_\tau = d_\tau + \frac{\Psi_\tau}{\mu_\tau \left(1 - Q_\tau^0\right)} \tag{7}$$

where

$$\Psi_\tau = \frac{\rho_\tau}{1-\rho_\tau} - \frac{(\beta+1)(\rho_\tau)^{\beta+1}}{1-(\rho_\tau)^{\beta+1}}$$
$$Q_\tau^0 = \frac{1-\rho_\tau}{1-(\rho_\tau)^{\beta+1}}$$
$$\rho_\tau = \frac{\lambda_\tau^c}{\mu_\tau}$$

d_τ is the total transmission time of input/output data to/from the CDC layer through MBS, and Q_τ^0 is the possibility when there are no tasks in the CDC layer in the current time slot.

In addition, to guarantee the stability of a task queue in the CDC layer, we have

$$\lambda_\tau^c \le \mu_\tau. \tag{8}$$

In time slot τ, $\widetilde{T}_\tau$ cannot exceed its limit T^c, that is

$$\widetilde{T}_\tau \le T^c. \tag{9}$$

C. Profit

Then, let F_τ, $\beth_\tau$, and Γ_τ denote total revenue, total cost, and total profit in time slot τ. Then

$$\Gamma_\tau = F_\tau - \beth_\tau. \tag{10}$$

To meet the actual response time requirements of users, we use service-level agreements (SLAs) [23] as legal contracts. They are typically predefined between users and cloud

providers. In this work, SLAs specify the revenue or penalty if the response time of tasks is within or beyond their limits, respectively. Long response time of tasks leads to worse quality of applications and it reduces the profit of the system providers that not only have to meet tasks' QoS but also to maximize the total profit. In this work, QoS is defined as the response time of tasks. We adopt the following SLA as an example. If the actual response time of a task is less than or equal to 0.1 s, its revenue is 0.5 \$; otherwise, its penalty is -0.2 \$. Let x denote the actual response time of a task and $f(x)$ denote its corresponding revenue or penalty. Then

$$f(x) = \begin{cases} 0.5, & x \leq 0.1 \text{ seconds} \\ -0.2, & x > 0.1 \text{ seconds}. \end{cases} \tag{11}$$

According to (11), the revenue brought by each task scheduled to execute in the edge computing layer is $f(\hat{T}_\tau)$. Then, the total revenue brought by all tasks scheduled to execute in the edge computing layer is $f(\hat{T}_\tau)I_\tau$. Similarly, the revenue brought by each task scheduled to execute in the CDC layer is $f(\widetilde{T}_\tau)$. Then, the revenue brought by all tasks scheduled to execute in CDC is $f(\widetilde{T}_\tau)(\lambda_\tau\gamma - I_\tau)$. F_τ is obtained as

$$F_\tau = f(\hat{T}_\tau)I_\tau + f(\widetilde{T}_\tau)(\lambda_\tau\gamma - I_\tau). \tag{12}$$

Let $\beth_\tau^E$ denote the execution cost of all tasks scheduled to execute in the edge computing layer in time slot τ and $\beth_\tau^C$ denote the energy cost of all tasks scheduled to execute in the CDC layer in time slot τ. Then, $\beth_\tau$ consists of $\beth_\tau^E$ and $\beth_\tau^C$. Let $\eta_\tau^{i,j}$ denote the execution cost of task i scheduled to execute in node j in the edge computing layer in time slot τ. Let $\widehat{\eta}$ denote the upper limit of $\eta_\tau^{i,j}$

$$\eta_\tau^{i,j} \leq \widehat{\eta} \tag{13}$$

$\beth_\tau^E$ is obtained as $\sum_{i=1}^{I_\tau}\sum_{j=1}^{J}(o_\tau^{i,j}\eta_\tau^{i,j})$. Let e_τ denote the price of power grid in time slot τ and E_τ denote the amount of energy consumed by all tasks scheduled to execute in the CDC layer in time slot τ. Thus, $\beth_\tau^E = e_\tau E_\tau$. Similar to [24] and [25], the data transmission cost between CDC and edge computing layers is ignored. Thus, $\beth_\tau$ is obtained as

$$\beth_\tau = e_\tau E_\tau + \sum_{i=1}^{I_\tau}\sum_{j=1}^{J} o_\tau^{i,j}\eta_\tau^{i,j}. \tag{14}$$

D. Energy Consumption

$\widehat{P}$ and $\widetilde{P}$ denote peak and idle power of each server in CDC layer, respectively. σ denotes the number of tasks processed by each powered-ON server per time in the CDC layer, and χ denotes the power usage effectiveness value [26] of CDC. Following [27], energy consumption E_τ is calculated as follows:

$$\begin{aligned} E_\tau &= \frac{\nu\mu_\tau + \psi\lambda_\tau^c(1 - q(\lambda_\tau^c, \mu_\tau))}{\sigma}\gamma \\ \nu &= \widetilde{P} + (\chi - 1)\widehat{P} \\ \psi &= \widehat{P} - \widetilde{P} \\ \delta(\lambda_\tau^c, \mu_\tau) &= \frac{1 - \rho_\tau}{1 - (\rho_\tau)^{\beta+1}}(\rho_\tau)^\beta. \end{aligned} \tag{15}$$

Let Δ denote the maximum amount of the total available energy in the CDC layer. Then, E_τ cannot exceed Δ, that is

$$\frac{\nu\mu_\tau + \psi\lambda_\tau^c(1 - q(\lambda_\tau^c, \mu_\tau))}{\sigma}\gamma \leq \Delta. \tag{16}$$

The execution cost of all tasks scheduled to execute in the edge computing layer needs to be less than or equal to the energy cost of all tasks scheduled to execute in the CDC layer in time slot τ, that is

$$\sum_{i=1}^{I_\tau}\sum_{j=1}^{J} o_\tau^{i,j}\eta_\tau^{i,j} \leq e_\tau E_\tau. \tag{17}$$

Let ϖ denote the maximum number of servers in the CDC layer. Then, the number of powered-ON servers is μ_τ/σ, which must satisfy:

$$\frac{\mu_\tau}{\sigma} \leq \varpi. \tag{18}$$

E. Load Balance Model

In the edge computing layer, several load factors can be used to evaluate the load balance of nodes. This work considers three important factors, including CPU, memory, and bandwidth [28]. To allocate resource reasonably, this work defines the load level of each node j in time slot τ as follows:

$$L_\tau^j = \alpha_1 C_\tau^{u,j} + \alpha_2 M_\tau^{u,j} + \alpha_3 B_\tau^{u,j} \tag{19}$$

where $C_\tau^{u,j}$, $M_\tau^{u,j}$, and $B_\tau^{u,j}$ denote the utilization of CPU, memory, and bandwidth resources of node j in time slot τ, respectively. α_1–α_3 denote their weights and satisfy

$$\alpha_1 + \alpha_2 + \alpha_3 = 1 \tag{20}$$

$C_\tau^{u,j}$, $M_\tau^{u,j}$, and $B_\tau^{u,j}$ cannot exceed 1, that is

$$C_\tau^{u,j} = \frac{\sum_{i=1}^{I_\tau} o_\tau^{i,j}\varsigma_i}{\widetilde{C}} \leq 1 \tag{21}$$

$$M_\tau^{u,j} = \frac{\sum_{i=1}^{I_\tau} o_\tau^{i,j}\varphi_i}{\widetilde{M}} \leq 1 \tag{22}$$

$$B_\tau^{u,j} = \frac{\sum_{i=1}^{I_\tau} o_\tau^{i,j}\xi_i}{\widetilde{B}} \leq 1 \tag{23}$$

where ς_i, φ_i, and ξ_i denote the amount of CPU, memory, and bandwidth resources needed by task i, respectively, and $\widetilde{C}$, $\widetilde{M}$, and $\widetilde{B}$ are the maximum available amount of CPU, memory, and bandwidth resources in node j, respectively.

Let $\overline{L}_\tau$ denote the average load level of all nodes in the edge computing layer in time slot τ. Then, we have

$$\overline{L}_\tau = \frac{\sum_{j=1}^{J} L_\tau^j}{J}. \tag{24}$$

Let ϕ_τ denote the load balance level of all nodes in the edge computing layer in time slot τ. We use the standard deviation to characterize it, that is

$$\phi_\tau = \sqrt{\frac{1}{J}\sum_{j=1}^{J}(L_\tau^j - \overline{L}_\tau)^2} \leq \widehat{L} \tag{25}$$

where $\widehat{L}$ denotes the specified maximum load balance level. It is worth noting that the lower ϕ_τ, the more balanced the edge computing layer.

F. Profit Maximization Problem

The objective is to maximize Γ_τ, that is

$$\underset{o_\tau^{i,j}, I_\tau, \mu_\tau}{\textbf{Max}} \ \{\Gamma_\tau\} \tag{26}$$

subject to (1)–(3), (6)–(9), (13), (16)–(18), (21)–(23), (25), and 27.

$$I_\tau \in N^+, \ \mu_\tau \geq 0, \ o_\tau^{i,j} \in \{0, 1\}. \tag{27}$$

Then, SMBO used to solve the problem is described in Section IV.

IV. Simulated-Annealing-Based Migrating Birds Optimization)

Note that Γ_τ is nonlinear in terms of $o_\tau^{i,j}$, I_τ, and μ_τ. Thus, it is a constrained nonlinear optimization problem. To well solve it, this work designs a method of penalty function to convert it into an unconstrained one, that is

$$\underset{o_\tau^{i,j}, I_\tau, \mu_\tau}{\textbf{Max}} \ \{\tilde{\Gamma}(\vec{\vartheta}) = \Gamma_\tau - \zeta \beth\}. \tag{28}$$

In (28), $\tilde{\Gamma}$ is the new objective function and ζ denotes a large positive constant. $\vec{\vartheta}$ is a decision vector, including $o_\tau^{i,j}$, I_τ, and μ_τ. $\beth$ denotes the penalty if any equality and inequality constraint is violated, that is

$$\beth = \sum_{y=1}^{Y} |w_y(\vec{\vartheta})|^2 + \sum_{z=1}^{Z} (\max\{0, -u_z(\vec{\vartheta})\})^2. \tag{29}$$

An equality constraint y is converted into $w_y(\vec{\vartheta}) = 0$. Its penalty is $\left|w_y(\vec{\vartheta})\right|^2$ if it is violated; otherwise, its penalty is 0. An inequality constraint z is transformed into $u_z(\vec{\vartheta}) \geq 0$. Its penalty is $(\max\{0, -u_z(\vec{\vartheta})\})^2$ if it is violated; otherwise, its penalty is 0.

There are several traditional methods, e.g., convex programming, conjugate gradient methods, quasi-Newton methods, and interior-point methods, to solve it. However, they often require the derivatives of objective functions in their optimization problems. Thus, they are applicable to solve some typical constrained optimization problems with specific mathematical characteristics [29]. In addition, their optimization steps are complicated and their obtained solutions to often intractable problems are usually of poor quality. To avoid such drawbacks of traditional algorithms, several nature-inspired metaheuristic-based optimization algorithms, e.g., tabu search, genetic algorithm (GA), and PSO, have been used to find the near-optimal solutions to large-scale constrained optimization problems. Each metaheuristic algorithm aims to find an improving solution iteration by iteration.

Among many, an emerging nature-inspired metaheuristic algorithm named migrating birds optimization (MBO) is proposed in [30]. It differs from other metaheuristic algorithms because it has a benefit mechanism among solutions. It is inspired and derived from an effective V flight formation of migrating birds (solutions), which brings benefits in energy saving because birds in other positions get benefit from the birds in front. In MBO, there is a leader bird in the flock (population) and two lines of other birds follow it. It is typically assumed that if the leader bird becomes tired after flying for a certain time, it flies to the end of a line and one of the other following birds becomes a new leader.

Each solution compares itself with a number of its own neighbors and several best neighbors of the front solution. It is replaced by the best of them if it becomes worse. It means that if one solution is not improved by its own neighbors, it may be replaced by one of neighbors of its front solution. This operation is repeated several times. Then, the leader solution goes to the last position, and one of the second solutions goes to the first position. The algorithm is terminated after a given number of iterations. MBO explores more areas within neighbor spaces of feasible solutions by first initializing a number of parallel solutions for birds in a V-formation. The leader bird in the V-formation spends the most energy. In MBO, the neighborhood within more promising solutions is explored in more detail. After several iterations, these solutions might move to different directions if they are improved along their ways. However, finally, most solutions of MBO may converge to one or several local optima.

Simulated annealing (SA) can conditionally jump from local optima by accepting some worse moves with the metropolis acceptance rule [31]. It is pointed out that SA can obtain global optima with large probability in theory, and SA is widely used to find high-quality solutions to different optimization problems. However, its main drawback is its very slow convergence speed. In this work, a hybrid algorithm called SMBO is proposed by combing SA's Metropolis acceptance rule and MBO. Specifically, this work adopts the SA-based update mechanism to change a solution. Its novelty includes the integration of an SA-based update mechanism into MBO and the disruptive selection of solutions to improve its solution diversity.

A. Solution (Bird) Encoding

Let h denote the number of solutions in the population. Each solution j ($1 \leq j \leq h$) includes decision variables, including $o_\tau^{i,j}$, I_τ, and μ_τ. $\vec{\vartheta}_j$ denotes the position of each solution j

$$\vec{\vartheta}_j = [o_\tau^{1,1}, \cdots \text{and } o_\tau^{I_\tau, J}, I_\tau, \mu_\tau]. \tag{30}$$

B. SA-Based Update Mechanism

This work adopts an SA-based update mechanism to change a solution j. Let $\vec{\vartheta}_j^{\,g}$ and $\vec{\vartheta}_j^{\,g+1}$ denote positions of each solution j in iterations g and $g+1$, respectively. If $\tilde{\Gamma}(\vec{\vartheta}_j^{\,g}) \geq \tilde{\Gamma}(\vec{\vartheta}_j^{\,g+1})$, $\vec{\vartheta}_j^{\,g+1}$ is selected to update solution j; otherwise, it is accepted only if

$$e^{\tilde{\Gamma}(\vec{\vartheta}_j^{\,g}) - \tilde{\Gamma}(\vec{\vartheta}_j^{\,g+1})} / t_g > \nu \tag{31}$$

where t_g is the current temperature in iteration g and ν is a random number in (0, 1).

C. Disruptive Selection

Disruptive selection chooses higher and lower quality solutions. First, χ_l is defined as the absolute difference between fitness value ($\tilde{\Gamma}_j$) of solution j and the average value ($\widetilde{\Gamma}$) for

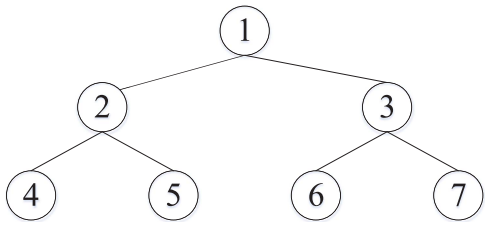

Fig. 2. Seven-solution V-formation.

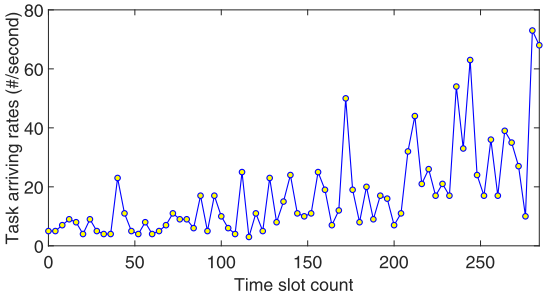

Fig. 3. Task arriving rate.

all solutions in the flock. Second, the new fitness function ($\widetilde{f}_j$) for solution j is obtained as $\chi_j / \sum_{j=1}^{h} \chi_j$. According to (32), decentralized solutions that are farther from the average value for all birds have a larger chance to be selected. Therefore, the disruptive selection can improve the diversity of birds in the flock by selecting diverse ones. $\widetilde{f}_j$ is given as

$$\widetilde{f}_j = \frac{\chi_j}{\sum_{j=1}^{h} \chi_j}$$
$$\chi_j = |\tilde{\Gamma}_j - \widetilde{\Gamma}|. \tag{32}$$

Let k denote the number of neighbor solutions for each solution and x denote the number of unused neighbor solutions of the current solution, which are shared with the next solution in the V-formation. Let m denote the number of times and G denote the number of total iterations. Algorithm 1 shows the details of SMBO. Line 1 initializes the parameters, including h, k, x, m, and G. Line 2 initializes a population of h initial solutions randomly. Line 3 sorts the initial flock according to the ascending order of their fitness values obtained with (28). In addition, at each iteration, solutions in the current flock are similarly sorted. Line 4 organizes them in a V-formation according to Fig. 2. Fig. 2 shows an exemplar flock, including seven solutions, and their V-formation. Here, solution 1 is the best, whereas solution 7 is the worst. Thus, solution 1 is corresponding to the leader bird, and its successor solutions are 2 and 3, respectively. It is worth noting that solution 3 is worse than solution 2, and both of them are worse than solution 1 and better than solutions 4–7.

Line 6 initializes t_1 with the starting temperature $\hat{t}$. Lines 9–30 perform m times to update all solutions in the flock. Line 10 creates k neighbors for the leader solution and sorts them. Line 11 increases g by k. Lines 12–15 create $k - x$ neighbors for each solution j ($2 \leq j \leq h$) and sort them. g is increased by $k - x$ for each solution. Line 16 moves the best neighbor of the leader solution to $\Theta(1)$, and puts the second and third-best neighbors of the leader solution into the neighbor sets of two successors of the leader solution. Lines 17–19 move the best neighbor of solution j ($2 \leq j \leq h-2$) to $\Theta(j)$, and put x unused best neighbors of solution j into the neighbor set of solution $j+2$. Line 20 moves the best neighbor of solution $h-1$ to $\Theta(h-1)$ and puts x unused best neighbors of solution $h-1$ into the neighbor set of solution $h-1$. Line 21 moves the best neighbor of solution h to $\Theta(h)$ and puts x unused best neighbors of solution h into the neighbor set of solution h. Lines 22–28 use the SA-based update mechanism to change solution j ($1 \leq j \leq h$) via (31). Specifically, if $\Theta(j)$ is better than solution j, solution j is updated with $\Theta(j)$; otherwise, the SA's metropolis rule is performed to update solution j via (31). Line 31 moves the leader solution to the end and forwards its left or right successor to the leader position. Line 32 sorts all solutions except the leader in the flock.

Let Θ denote the set of best solutions in the flock. Line 31 moves the leader solution to the end and forwards its left or right successor to the leader position. Specifically, the leader solution is alternately moved to the left or right end. For example, if the leader solution in the current iteration is moved to the left end, the left successor is forwarded to the leader position. Then, the leader solution in the next iteration is moved to the right end, and the right successor is forwarded to the leader position. Line 32 sorts all solutions except the leader in the flock. Line 34 reduces temperature t_g by ε, which denotes the temperature cooling rate. The **while** loop stops if $g = G$. Line 36 returns the best solution (leader bird) in the flock, which is converted into decision variables, including $[o_\tau^{1,1}, \ldots,$ and $o_\tau^{I_\tau,J}, I_\tau, \mu_\tau]$.

V. Performance Evaluation

This work evaluates the proposed SMBO with real-life data. SMBO is implemented with MATLAB 2017 in a computer with an Intel Xeon CPU with 2.4 GHz and a 32-GB memory.

A. Parameter Setting

This work uses realistic tasks collected from Google cluster trace[1] to evaluate the proposed method. Fig. 3 shows the task arriving rate in one day. In addition, this work uses the realistic price of power grid collected from the New York Independent System Operator.[2] Fig. 4 shows the price of power grid in one day. Besides, the length of one time slot is 300 s, i.e., $\gamma = 300$ s.

In addition, the following parameters are set as follows [32]. $\widehat{L} = 0.01$, $J = 30$, $\widetilde{B} = 3000$ MB/s, $\widetilde{M} = 1024$ MB, and $\widetilde{C} = 2048$ MIPS. In addition, ξ_i, φ_i, and ς_i are randomly produced in the range of (0, 0.1), i.e., $\xi_i \in$ (0,0.1) MB/s, $\varphi_i \in$ (0, 0.1) MB, and $\varsigma_i \in$ (0, 0.1) MIPS. In addition, $\sigma = 0.05$ tasks/s, $\varpi = 2 \times 10^3$, $\Delta = 1 \times 10^5$ WH, $\widehat{P} = 600$ W, $\widetilde{P} = 300$ W, $\chi = 1.6$, and $\widehat{\eta} = 0.01$ \$. Besides, d_τ is set to $(1/5)\Psi_\tau/(\mu_\tau(1-Q_\tau^0))$, i.e., $d_\tau = (1/5)\Psi_\tau/(\mu_\tau(1-Q_\tau^0))$ seconds. In addition, $\eta_\tau^{i,j}$ is randomly produced in the range of

[1]https://github.com/google/cluster-data
[2]https://www.nyiso.com/

Algorithm 1 SMBO

Initialize parameters including h, k, x, m and G
Initialize a population of h initial solutions randomly
Sort the initial flock according to their fitness values obtained with (28)
Organize them in a V-formation according to Fig. 2
$g \leftarrow 1$
$t_g \leftarrow \hat{t}$
while $g \leq G$ **do**
  $i \leftarrow 1$
  while $i \leq m$ **do**
    Create k neighbors for the leader solution and sort them
    $g = g + k$
    for $j \leftarrow 2$ to h **do**
      Create $k - x$ neighbors for each solution j and sort them
      $g = g + k - x$
    end for
    Move the best neighbor of the leader solution to $\Theta(1)$, and put the second-best and the third-best neighbors of the leader solution into the neighbor sets of two successors of the leader solution
    for $j \leftarrow 2$ to $h - 2$ **do**
      Move the best neighbor of solution j to $\Theta(j)$, and put x unused best neighbors of solution j into the neighbor set of solution $j + 2$
    end for
    Move the best neighbor of solution $h - 1$ to $\Theta(h - 1)$, and put x unused best neighbors of solution $h - 1$ into the neighbor set of solution $h - 1$
    Move the best neighbor of solution h to $\Theta(h)$, and put x unused best neighbors of solution h into the neighbor set of solution h
    for $j \leftarrow 1$ to h **do**
      if $\Theta(j)$ is better than solution j **then**
        Update solution j with $\Theta(j)$
      else
        Use SA's metropolis rule to update solution j with (31)
      end if
    end for
    $i \leftarrow i + 1$
  end while
  Move the leader solution to the end and forward its left successor or right successor to the leader position
  Sort all solutions except the leader in the flock
  $g \leftarrow g + 1$
  $t_g \leftarrow t_{g-1} * \varepsilon$
end while
Return the best solution (leader bird) in the flock

(0, 0.01), i.e., $\eta_\tau^{i,j} \in (0, 0.01)$ \$. $T^c = 0.1$ s, $T^+ = 0.1$ s, $\gamma = 300$ s, $\beta = 30$, $n_\tau^j = 1000$, $\alpha_1 = 0.3333$, $\alpha_2 = 0.3333$, and $\alpha_3 = 0.3333$. Besides, k_i is randomly produced in the range of (1×10^6 and 8×10^6) processing operations, i.e., $k_i \in (1 \times 10^6$ and $8 \times 10^6)$. p_j is randomly produced in

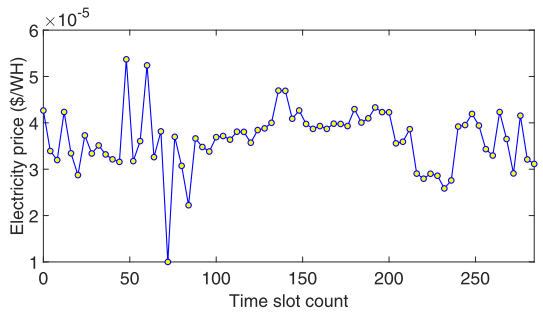

Fig. 4. Price of power grid in one day.

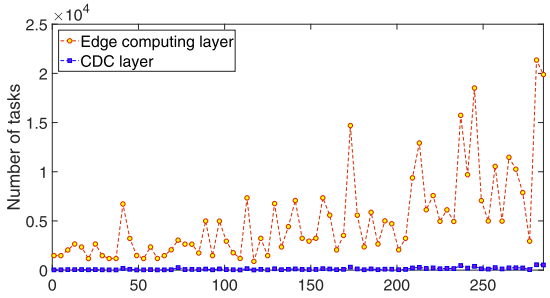

Fig. 5. Number of tasks scheduled to edge computing and CDC layers.

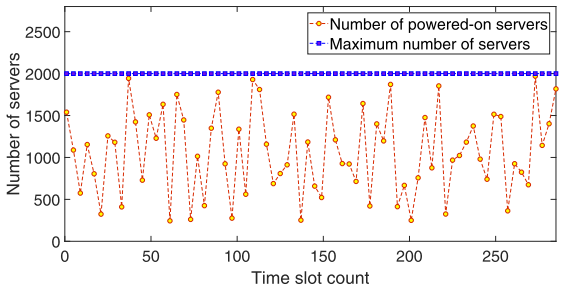

Fig. 6. Number of powered-on servers in the CDC layer.

the range of (1×10^{11} and 2×10^{11}) processing operations per second, i.e., $k_i \in (1 \times 10^{11}$ and $2 \times 10^{11})$. The parameters of SMBO are set as follows: $h = 51$, $k = 3$, $x = 1$, $m = 10$, $G = 51^3$, $\hat{t} = 5 \times 10^6$, and $\varepsilon = 0.985$. In addition, $\zeta = 10^{10}$.

B. Experimental Results

Fig. 5 shows the number of tasks scheduled to edge computing, i.e., local computing, and CDC layers. It is shown that the number of tasks scheduled to the CDC layer is much lower than that scheduled to the edge layer in each time slot. The reason is that nodes in the edge layer are much closer to users and can avoid the transmission delay of input/output data to/from the CDC layer through MBS. Since CPU, memory, bandwidth, and storage resources, available energy, and the load balance of all heterogeneous nodes in the edge computing layer are all limited, therefore, a few tasks also need to be offloaded to execute in servers in the CDC layer, but the number of offloaded tasks is much lower than that executed in nodes in the edge layer. Fig. 6 shows the number of powered-ON servers in the CDC layer. The maximum number of servers

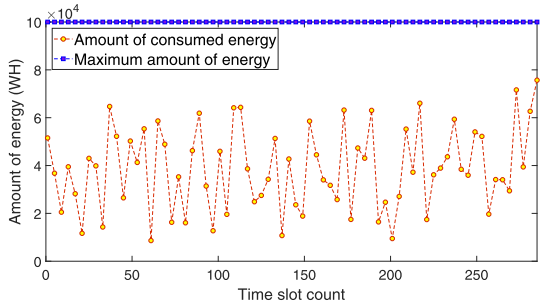

Fig. 7. Amount of energy consumed in the CDC layer.

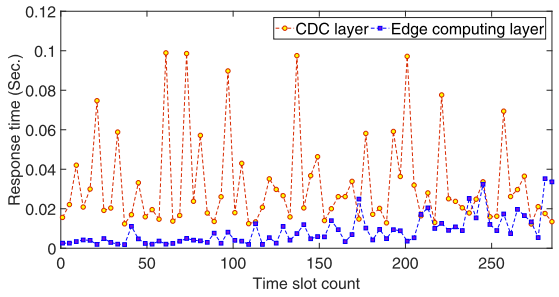

Fig. 8. Total response time of each task in CDC and edge computing layers.

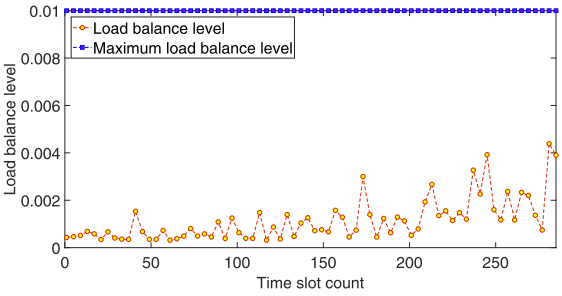

Fig. 9. Load balance level of all nodes in edge computing layer.

in the CDC layer is 2000, i.e., $\varpi = 2000$. It is shown that the number of powered-ON servers in the CDC layer is less than its limit in each time slot.

Fig. 7 shows the amount of energy consumed in CDC in each time slot. The maximum amount of the total available energy in the CDC layer is 1×10^5 WH, i.e., $\Delta = 1 \times 10^5$ WH. It is shown that the amount of energy consumed in the CDC layer is less than its limit in each time slot. Fig. 8 shows the total response time of each task executed in CDC and edge computing layers. The users' response time limits for each task scheduled to execute in the edge computing and CDC layers are both set to 0.1 s, i.e., $T^+ = T^c = 0.1$ s. Therefore, it is shown that the total response time of each task executed in CDC and edge computing layers is less than its limit in each time slot, i.e., (6) and (9) are both met in each time slot. The results demonstrate that the execution results based on our obtained schedule can strictly meet the response time limits of tasks.

Fig. 9 shows the load balance level of all nodes in the edge computing layer in each time slot. The specified maximum load balance level in the edge computing layer is 0.01, i.e., $\widehat{L} = 0.01$. It is shown that the load balance level of all nodes in the edge computing layer is less than its limit in each time slot. This result demonstrates that SMBO provides the load balance among nodes in the edge computing layer. The reason is that SMBO jointly considers CPU, memory, and bandwidth resource limits, the load balance needs of all nodes, and different processing capacities of heterogeneous nodes in the edge computing layer.

C. Comparison Results

To validate SMBO, this work compares it with two typical metaheuristic optimization algorithms, including firefly algorithm (FA) [33] and genetic learning particle swarm optimization (GL-PSO) [34], and two baseline algorithms, including local computing and entire offloading. Here, SMBO, FA, and GL-PSO are repeated independently for 30 times to produce their respective results. The reasons for choosing them as benchmark algorithms are given as follows.

1) *FA [33]:* As an emerging metaheuristic algorithm, FA can efficiently find high-quality optima of multimode functions. Each of its fireflies is independent of each other, and it is easy to be implemented in parallel. Its convergence speed is fast, but it suffers from a premature convergence problem. The oscillation in its search process happens repeatedly, and its accuracy is often unsatisfying for high-dimensional optimization problems.
2) *GL-PSO [34]:* GL-PSO applies a genetic learning mechanism of a GA to construct exemplars that are hybridized with PSO in a cascade manner. Then, particles in PSO are guided by the exemplars produced by GA. Thus, GA and PSO are integrated cohesively to diversify the search of particles, thus guiding the population to find high-quality solutions. The search history of particles in PSO provides promising genetic information to GA and helps it reproduce high-quality exemplars.
3) *Local Computing:* All tasks of users are executed by nodes in the edge computing layer.
4) *Entire Offloading:* All tasks of users are offloaded to servers in the CDC layer.

The comparison between SMBO and FA can demonstrate the former's convergence speed. The comparison between it and GL-PSO can demonstrate its solution accuracy. Besides, SMBO, FA, and GL-PSO are all sensitive to the setting of their parameters. Therefore, similar to SMBO, several trials are conducted to determine the parameter setting for both FA and GL-PSO with the grid search method [35]. Besides, SMBO, FA, and GL-PSO stop their search processes if their solutions are not improved in ten consecutive iterations.

Fig. 10 shows the profit computed at each iteration of SMBO, FA, and GL-PSO in time slot 1. Each iteration of SMBO refers to Lines 8–34 in Algorithm 1. The iterations of FA and GL-PSO are similar to that of SMBO. FA and GL-PSO require 148 and 195 iterations to stop their searches, and their final profits are 365.72 $ and 694.76 $, respectively. On the other hand, SMBO only requires 126 iterations to stop its search, and its final profit is 734.35 $. It is observed that compared with FA and GL-PSO, SMBO's average profit

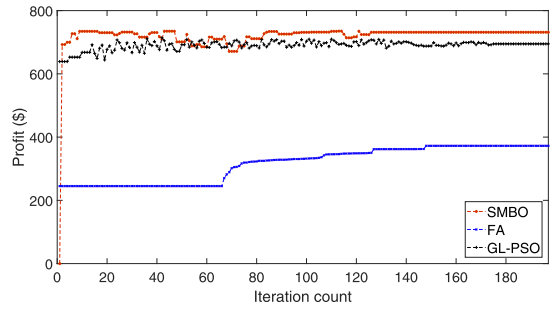

Fig. 10. Profit of each iteration of SMBO, FA, and GL-PSO in time slot 1.

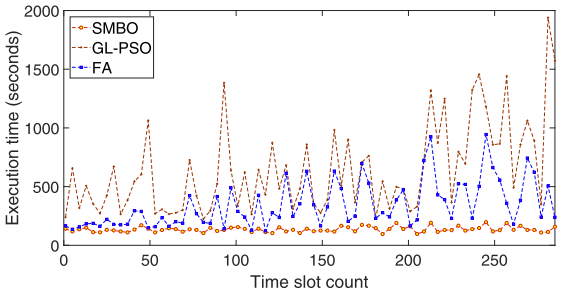

Fig. 11. Convergence time comparison of SMBO, FA, and GL-PSO.

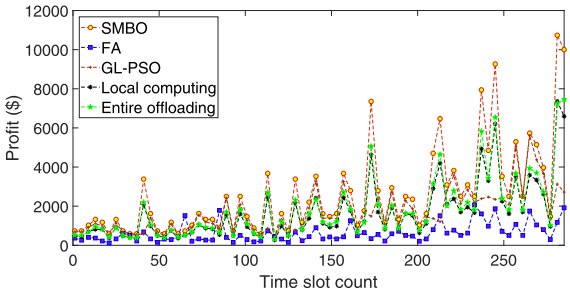

Fig. 12. Profit comparison of SMBO, FA, GL-PSO, local computing, and entire offloading.

of all time slots is increased by 50.20% and 5.39% on average, respectively. Fig. 11 shows the convergence time comparison of SMBO, FA, and GL-PSO in each time slot. It is shown that compared with FA and GL-PSO, SMBO's average convergence time of all time slots is reduced by 49.26% and 72.21% on average, respectively. Therefore, SMBO increases the profit in less time and fewer iterations than FA and GL-PSO. Figs. 10 and 11 prove that the adoption of SA's metropolis acceptance rule in the proposed method increases the diversity of population and search performance.

Fig. 12 shows the profit comparison of SMBO, FA, GL-PSO, local computing, and entire offloading. It is shown that compared with FA, GL-PSO, local computing, and entire offloading, the profit of SMBO is increased by 66.83%, 21.32%, 34.81%, and 30.22% on average, respectively. The reasons are given as follows. As shown in Fig. 10, the solution accuracy of SMBO is higher than those of FA and GL-PSO, and therefore, the profit of SMBO is larger than those of FA and GL-PSO. Local computing schedules all tasks of users to execute in nodes of the edge computing layer. It is worth noting that battery energy, CPU, memory, and bandwidth resources are all limited in nodes in the edge computing layer. Similarly, entire offloading schedules all tasks of users to execute in servers of the CDC layer. However, the maximum amount of energy, the maximum number of servers, and so on in the CDC layer are also limited. Thus, tasks scheduled with local computing and entire offloading need to wait for execution and suffer from higher latency, resulting in bad user experience and low profit. The experimental results validate that the proposed offloading method in SMBO outperforms these four benchmark methods on profit. This is because the proposed method jointly considers computation offloading between CDC and the edge, and CPU, memory, and bandwidth allocation to nodes in the edge computing layer.

VI. Conclusion

In recent years, edge computing has been gaining enormous attention and is glowingly adopted due to its fast response and close proximity to users when computing tasks are not intensive. However, its computing resources and energy capacity are limited. CDCs have rich computation resources but are usually located in remote sites and need both energy and time to transmit programs or data to CDCs and retrieve results from CDCs. Consequently, offloading tasks in the edge to CDCs is becoming a promising alternative to maximize the profit while enforcing user-specific response time limits of tasks. Based on an architecture including terminal, edge computing, and CDC layers, this work proposes a profit-maximized collaborative computation offloading and resource allocation algorithm. It jointly optimizes the computation offloading between CDC and edge computing layers and specifies resource allocation in CDC layer. In each time slot, a single-objective constrained optimization problem is formulated and solved by a proposed SMBO algorithm. Realistic data-driven experimental results demonstrate that SMBO obtains a larger profit than its several existing methods.

References

[1] M. Saez, F. P. Maturana, K. Barton, and D. M. Tilbury, "Real-time manufacturing machine and system performance monitoring using Internet of Things," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 4, pp. 1735–1748, Oct. 2018.

[2] J. Bi *et al.*, "Application-aware dynamic fine-grained resource provisioning in a virtualized cloud data center," *IEEE Trans. Autom. Sci. Eng.*, vol. 14, no. 2, pp. 1172–1184, Apr. 2017.

[3] H. Yuan, J. Bi, W. Tan, and B. H. Li, "Temporal task scheduling with constrained service delay for profit maximization in hybrid clouds," *IEEE Trans. Autom. Sci. Eng.*, vol. 14, no. 1, pp. 337–348, Jan. 2017.

[4] L. Liu, Z. Chang, X. Guo, S. Mao, and T. Ristaniemi, "Multiobjective optimization for computation offloading in fog computing," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 283–294, Feb. 2018.

[5] H. Yuan, J. Bi, W. Tan, M. Zhou, B. H. Li, and J. Li, "TTSA: An effective scheduling approach for delay bounded tasks in hybrid clouds," *IEEE Trans. Cybern.*, vol. 47, no. 11, pp. 3658–3668, Nov. 2017.

[6] Y.-D. Lin, E. T.-H. Chu, Y.-C. Lai, and T.-J. Huang, "Time-and-energy-aware computation offloading in handheld devices to coprocessors and clouds," *IEEE Syst. J.*, vol. 9, no. 2, pp. 393–405, Jun. 2015.

[7] M. Gaggero and L. Caviglione, "Model predictive control for energy-efficient, quality-aware, and secure virtual machine placement," *IEEE Trans. Autom. Sci. Eng.*, vol. 16, no. 1, pp. 420–432, Jan. 2019.

[8] X. T. R. Kong, S. X. Xu, M. Cheng, and G. Q. Huang, "IoT-enabled parking space sharing and allocation mechanisms," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 4, pp. 1654–1664, Oct. 2018.

[9] J. Zhao, Q. Li, Y. Gong, and K. Zhang, "Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 8, pp. 7944–7956, Aug. 2019.

[10] M. Liu and Y. Liu, "Price-based distributed offloading for mobile-edge computing with computation capacity constraints," *IEEE Wireless Commun. Lett.*, vol. 7, no. 3, pp. 420–423, Jun. 2018.

[11] Z. Wei, B. Zhao, J. Su, and X. Lu, "Dynamic edge computation offloading for Internet of Things with energy harvesting: A learning method," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4436–4447, Jun. 2019.

[12] J. Zheng, Y. Cai, Y. Wu, and X. Shen, "Dynamic computation offloading for mobile cloud computing: A stochastic game-theoretic approach," *IEEE Trans. Mobile Comput.*, vol. 18, no. 4, pp. 771–786, Apr. 2019.

[13] X. Tao, K. Ota, M. Dong, H. Qi, and K. Li, "Performance guaranteed computation offloading for mobile-edge cloud computing," *IEEE Wireless Commun. Lett.*, vol. 6, no. 6, pp. 774–777, Dec. 2017.

[14] C. Zhu *et al.*, "Folo: Latency and quality optimized task allocation in vehicular fog computing," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4150–4161, Jun. 2019.

[15] 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.

[16] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4005–4018, Jun. 2019.

[17] D. Xu, Q. Li, and H. Zhu, "Energy-saving computation offloading by joint data compression and resource allocation for mobile-edge computing," *IEEE Commun. Lett.*, vol. 23, no. 4, pp. 704–707, Apr. 2019.

[18] M. Li, F. R. Yu, P. Si, and Y. Zhang, "Energy-efficient machine-to-machine (M2M) communications in virtualized cellular networks with mobile edge computing (MEC)," *IEEE Trans. Mobile Comput.*, vol. 18, no. 7, pp. 1541–1555, Jul. 2019.

[19] G. Zhang, Y. Chen, Z. Shen, and L. Wang, "Distributed energy management for multiuser mobile-edge computing systems with energy harvesting devices and QoS constraints," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4035–4048, Jun. 2019.

[20] 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.

[21] J. Cao, K. Hwang, K. Li, and A. Y. Zomaya, "Optimal multiserver configuration for profit maximization in cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1087–1096, Jun. 2013.

[22] J. Yao, H. Guan, J. Luo, L. Rao, and X. Liu, "Adaptive power management through thermal aware workload balancing in Internet data centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 9, pp. 2400–2409, Sep. 2015.

[23] A. Barri and A. Dooms, "Data-driven modules for objective visual quality assessment focusing on benchmarking and SLAs," *IEEE J. Sel. Topics Signal Process.*, vol. 11, no. 1, pp. 196–205, Feb. 2017.

[24] O. Munoz, A. Pascual-Iserte, and J. Vidal, "Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading," *IEEE Trans. Veh. Technol.*, vol. 64, no. 10, pp. 4738–4755, Oct. 2015.

[25] 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. 4268–4282, Oct. 2016.

[26] J. Conejero, O. Rana, P. Burnap, J. Morgan, B. Caminero, and C. Carrión, "Analyzing Hadoop power consumption and impact on application QoS," *Future Gener. Comput. Syst.*, vol. 55, pp. 213–223, Feb. 2016.

[27] A. Kiani and N. Ansari, "Profit maximization for geographically dispersed green data centers," *IEEE Trans. Smart Grid*, vol. 9, no. 2, pp. 703–711, Mar. 2018.

[28] M. Hu, L. Zhuang, D. Wu, Y. Zhou, X. Chen, and L. Xiao, "Learning driven computation offloading for asymmetrically informed edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 8, pp. 1802–1815, Aug. 2019.

[29] Y. Du, C. Xu, and D. Tao, "Matrix factorization for collaborative budget allocation," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 4, pp. 1471–1482, Oct. 2018.

[30] E. Duman, M. Uysal, and A. F. Alkaya, "Migrating birds optimization: A new metaheuristic approach and its performance on quadratic assignment problem," *Inf. Sci.*, vol. 217, pp. 65–77, Dec. 2012.

[31] F. Javidrad and M. Nazari, "A new hybrid particle swarm and simulated annealing stochastic optimization method," *Appl. Soft Comput.*, vol. 60, pp. 634–654, Nov. 2017.

[32] H. Yuan, J. Bi, W. Tan, and B. H. Li, "CAWSAC: Cost-aware workload scheduling and admission control for distributed cloud data centers," *IEEE Trans. Autom. Sci. Eng.*, vol. 13, no. 2, pp. 976–985, Apr. 2016.

[33] K. Jagatheesan, B. Anand, S. Samanta, N. Dey, A. S. Ashour, and V. E. Balas, "Design of a proportional-integral-derivative controller for an automatic generation control of multi-area power thermal systems using firefly algorithm," *IEEE/CAA J. Automatica Sinica*, vol. 6, no. 2, pp. 503–515, Mar. 2019.

[34] Y.-J. Gong *et al.*, "Genetic learning particle swarm optimization," *IEEE Trans. Cybern.*, vol. 46, no. 10, pp. 2277–2290, Oct. 2016.

[35] F. J. Pontes, G. F. Amorim, P. P. Balestrassi, A. P. Paiva, and J. R. Ferreira, "Design of experiments and focused grid search for neural network parameter optimization," *Neurocomputing*, vol. 186, pp. 22–34, Apr. 2016.

Haitao Yuan (Member, IEEE) received the B.S. and M.S. degrees in software engineering from Northeastern University, Shenyang, China, in 2010 and 2012, respectively, the Ph.D. degree in modeling simulation theory and technology from Beihang University, Beijing, China, in 2016, and the Ph.D. degree in computer engineering from the New Jersey Institute of Technology (NJIT), Newark, NJ, USA, in 2020.

He was an Associate Professor with the School of Software Engineering, Beijing Jiaotong University, Beijing. He was a Ph.D. Student with the Department of Computer Science, City University of Hong Kong, Hong Kong, from 2013 to 2014. He was a Visiting Doctoral Student with NJIT in 2015. He has over 70 publications in international journals and conference proceedings, including the IEEE TRANSACTIONS ON CLOUD COMPUTING, the IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING, the IEEE TRANSACTIONS ON SERVICES COMPUTING, the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, and the IEEE TRANSACTIONS ON CYBERNETICS. His research interests include cloud computing, edge computing, data centers, big data, machine learning, deep learning, and optimization algorithms.

Dr. Yuan was a recipient of the 2011 Google Excellence Scholarship and the Best Paper Award-Finalist in the 16th IEEE International Conference on Networking, Sensing and Control.

MengChu Zhou (Fellow, IEEE) received the B.S. degree in control engineering from the Nanjing University of Science and Technology, Nanjing, China, in 1983, the M.S. degree in automatic control from the Beijing Institute of Technology, Beijing, China in 1986, and the Ph.D. degree in computer and systems engineering from the Rensselaer Polytechnic Institute, Troy, NY, USA, in 1990.

He joined the New Jersey Institute of Technology (NJIT), Newark, NJ, USA, in 1990, where he is currently a Distinguished Professor of Electrical and Computer Engineering. He has over 800 publications, including 12 books, more than 500 journal papers (more than 400 in IEEE TRANSACTIONS), 12 patents, and 28 book chapters. His research interests are in Petri nets, intelligent automation, Internet of Things, big data, web services, and intelligent transportation.

Dr. Zhou is a fellow of the International Federation of Automatic Control (IFAC), the American Association for the Advancement of Science (AAAS), and the Chinese Association of Automation (CAA). He is a Life Member of the Chinese Association for Science and Technology-USA, where he served as the President in 1999. He was a recipient of the Humboldt Research Award for U.S. Senior Scientists from the Alexander von Humboldt Foundation and the Franklin V. Taylor Memorial Award and the Norbert Wiener Award from the IEEE Systems, Man and Cybernetics Society. He is the Founding Editor of the IEEE Press Book Series on Systems Science and Engineering and the Editor-in-Chief of the IEEE/CAA JOURNAL OF AUTOMATICA SINICA.