Deadline-constrained Multi-resource Task Mapping and Allocation for Edge-Cloud Systems

Chuanchao Gao[1,2] , Aryaman Shaan[1], Arvind Easwaran[1,2]
[1]*School of Computer Science and Engineering*
[2]*Energy Research Institute @ NTU, Interdisciplinary Graduate Programme*
Nanyang Technological University
Singapore
gaoc0008@e.ntu.edu.sg, shaa0002@e.ntu.edu.sg, arvinde@ntu.edu.sg

GLOBECOM 2022 - 2022 IEEE Global Communications Conference | 978-1-6654-3540-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/GLOBECOM48099.2022.10001137

***Abstract*—In an edge-cloud system, mobile devices can offload their computation intensive tasks to an edge or cloud server to guarantee the quality of service or satisfy task deadline requirements. However, it is challenging to determine where tasks should be offloaded and processed, and how much network and computation resources should be allocated to them, such that a system with limited resources can obtain a maximum profit while meeting the deadlines. A key challenge in this problem is that the network and computation resources could be allocated on different servers, since the server to which a task is offloaded (e.g., a server with an access point) may be different from the server on which the task is eventually processed. To address this challenge, we first formulate the task mapping and resource allocation problem as a non-convex Mixed-Integer Nonlinear Programming (MINLP) problem, known as NP-hard. We then propose a zero-slack based greedy algorithm (ZSG) and a linear discretization method (LDM) to solve this MINLP problem. Experiment results with various synthetic tasksets show that ZSG has an average of 2.98% worse performance than LDM with a minimum unit of 5 but has an average of 6.88% better performance than LDM with a minimum unit of 15.**

***Index Terms*—multi-resource mapping and allocation, deadline requirements, edge-cloud computing**

I. Introduction

Compute intensive tasks are rapidly emerging with the development of Internet of Things and Artificial Intelligence technologies, and this coupled with the deadline requirements of time-critical tasks, introduce a big challenge for systems. For example, in autonomous driving applications, tasks such as object detection and localization fall into this category, and the vehicles (*end devices*) are required to service these tasks while meeting their deadlines. The multi-layer edge-cloud system is often deployed to enhance the end devices' capability of handling such tasks, and such capability will be increased further with the advent of wireless technologies that are capable in handling strict deadlines such as 5G-URLLC [1].

In a multi-layer edge-cloud system, tasks can be offloaded from end devices to *access points*, and then forwarded to *servers* for timely processing. Servers that are located far away from the end devices, which results in significant data transmission latency, are called cloud servers. Servers that are deployed collectively with access points to provide a quick response to end devices are called edge servers. The computation capacity of a cloud server is usually much greater than that of an edge server. If the tasks received by access points have significant demand for computation resource, the collectively deployed edge servers may not have enough computation resource to finish these tasks by their deadlines. In such a case, the access points can forward the tasks with high computation resource demand to cloud servers for processing.

End devices communicate with access points through a wireless network, and access points communicate with servers through a wired backhaul network that has a much larger bandwidth capacity than the wireless network. Due to the limited wireless bandwidth and computation resource, it is challenging to determine where the tasks should be offloaded and processed (*task mapping problem*), and how much bandwidth and computation resource should be allocated to them (*resource allocation problem*), to maximize system profit while meeting task deadlines. This problem is further compounded by the fact that the access point that a task is offloaded to and the server that it is eventually processed on may be deployed at different locations. An access point will allocate bandwidth to tasks that are offloaded to it, and a server will allocate computation resource to tasks that are processed on it.

In this paper, we formulate the above problem as a non-convex MINLP. Nonconvexity arises due to the deadline constraint, since the allocated wireless bandwidth (respectively computation) has an inverse relation to the time taken for offloading (respectively processing). In our model, end devices can offload tasks to one of several nearby access points using the allocated wireless bandwidth, and each task can be processed on any reachable server using the allocated computation resource. Besides, there is an additional transmission delay incurred by the task if the offloading access point and the processing server are deployed at different locations. This introduces further challenges because the end-to-end deadline now depends on three factors: 1) wireless bandwidth allocated by the offloading access point, 2) transmission delay between the offloading access point and the processing server, and 3) computation resource allocated by the processing server. A task mapping and resource allocation is deemed *feasible* in

This work was supported in part by the MoE Tier-2 grant MOE-T2EP20221-0006.

this model if the task can be completed by its deadline with the allocated bandwidth and computation resource, inclusive of any transmission delays.

This paper aims to maximize the total system profit, where each task can contribute to this profit only if its allocation is feasible. From the literature on knapsack problems [2], the above problem can be categorized as a Generalized Assignment Problem (GAP) with fixed profit and bin-specific sizes for each item, assuming either the wireless bandwidth or the computation resource allocation is fixed. The intuition is that in order to meet task deadlines, the allocation of bandwidth and computation resource depends on the access point and the server to which the task is mapped and the transmission delay between them. Note that GAP is known to be NP-Hard and more specifically APX-Hard [2]. The contributions of this paper are as follows.

- We formulate the deadline-constrained task mapping and resource allocation problem with communication and computation contention as a nonconvex MINLP.
- We propose a zero-slack based greedy heuristic algorithm (ZSG) for the above problem, and the resources allocated to all provisioned tasks are just enough for these tasks to be completed exactly at their respective deadlines. We also propose a linear discretization method (LDM) to reformulate the nonconvex MINLP problem into an Integer Linear Programming problem, assuming that the bandwidth and computation resources can only be allocated in discrete units.
- We conduct experiments with synthetically generated tasksets to evaluate the performance of the two proposed methods. Results show that ZSG can obtain 2.98% less profit than LDM with a minimum unit of 5 and 6.88% more profit than LDM with a minimum unit of 15. Further, the performance of LDM critically depends on the size of the minimum discrete unit that can be allocated; the achieved profit drops by 9.87% on an average when the minimum unit is increased from 5 to 15.

Related Work. Few studies have considered this deadline-constrained problem with both computation and communication contention, aimed at minimizing either the total system cost or energy consumption [3], [4]. However, these studies assumed that tasks could be directly offloaded to the servers where they are processed, and hence the multi-resource contention is modeled on the same server for each task. Other studies have considered similar deadline-constrained problems with either a fixed task to server mapping [5] or a fixed resource allocation for each task [6]. Finally, a task mapping and computation resource allocation problem with deadlines has also been considered [7], but this study assumes that the bandwidth allocated to all tasks for offloading is fixed. A recent survey provides a comprehensive list of studies that consider deadline-constrained problems under various settings [8]. Thus, to the best of our knowledge, there is no study in the literature that considers the problem setting of allocating varying bandwidth and computation resource to tasks by units (access points and servers) at different locations, while having an end-to-end deadline requirement.

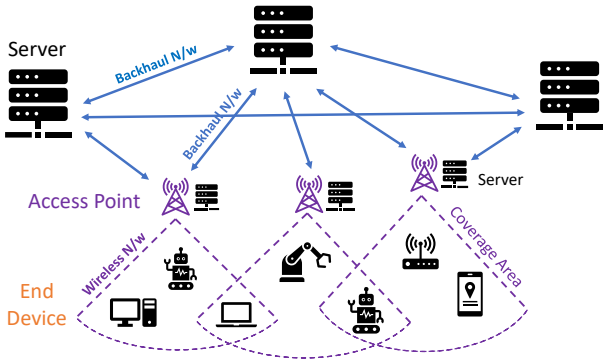

Fig. 1. Edge-Cloud System Model

II. System Model and Problem Formulation

A. Edge-Cloud System Model

The multi-layer edge-cloud system comprises end devices, access points and servers, as shown in Fig.1. *End devices* are machines that have specific functionalities and can communicate with access points through a wireless network. We denote the set of tasks generated by end devices as $\mathcal{I}$. Each task i, where $i \in \mathcal{I}$, has four associated parameters: $\{s_i, q_i, \Delta_i, p_i\}$. s_i is the amount of data to be offloaded by task i, q_i is the total number of CPU cycles required for task i, Δ_i is the task end-to-end deadline, and p_i is the profit gained by completing task i before its deadline. If task i misses its deadline, the system will not get any profit from this task. Besides, we assume that a task cannot be split, *i.e.*, it must be entirely offloaded to one access point and processed on one server. We also assume that the tasks can only be processed on servers, and therefore must necessarily be offloaded to derive profit.

Access Points are located near end devices and can collect tasks from end devices within their coverage area through wireless communication. After receiving tasks from end devices, access points will forward these tasks to servers for processing through the backhaul network. We denote the access point set as $\mathcal{A}$. Each task is covered only by a subset of access points, and we denote the access point subset that supports task i offloading as $\mathcal{A}_i$. The wireless bandwidth capacity of an access point $j \in \mathcal{A}$ is denoted as b_j.

Servers are units that process tasks forwarded from access points through a backhaul network. Compared with end devices, servers have a much greater computation resource capacity and can assist the end devices in processing computation intensive tasks. Note that the servers that are deployed collectively with access points are called edge servers, and the servers that are deployed far away from access points are called cloud servers. Compared with cloud servers, edge servers can provide a quicker response to end devices, but usually have a smaller computation resource capacity. We denote the server set as $\mathcal{N}$. The computation resource capacity of a server $k \in \mathcal{N}$ is denoted as c_k.

We assume that the backhaul network between access points and servers is wired, and therefore can support data transmission with constant delay irrespective of the data size, *i.e.*, the available bandwidth in this network is sufficiently large. We denote the transmission delay between an access point j and a server k as δ_{jk}, where $\delta_{jk} = \delta_{kj}$ and $\delta_{jk} = 0$ when access point j and server k are collectively deployed.

B. Problem Formulation

We use a binary variable x_{ij} to denote the offloading decision for task i. $x_{ij} = 1$ if and only if task i is offloaded to access point $j \in \mathcal{A}_i$. Similarly, we use a binary variable y_{ik} to denote the processing decision of task i. $y_{ik} = 1$ if and only if task i is processed on server $k \in \mathcal{N}$. Furthermore, we use the variable b_{ij} to denote the amount of wireless bandwidth that will be assigned to task i by access point j, and the variable c_{ik} to denote the amount of computation resource that will be assigned to task i by server k. A summary of the notation used in this paper is provided in Table I.

The total time taken to complete a task i, denoted as T_i, consists of four parts: task offloading time T_i^o, data transmission time in the backhaul network T_i^c, task processing time T_i^p, and the result return time. Normally, the data size of the result is negligible, so the time spent for result downloading from access points to end devices is assumed constant, and is deducted from the task deadline. In other words, the result return time is assumed to be equal to T_i^c after the result downloading time is deducted from the task deadline. If task i is offloaded to access point j and processed on server k, $T_i^o = s_i/b_{ij}$, $T_i^c = \delta_{jk}$, and $T_i^p = q_i/c_{ik}$. Thus, the total time taken to complete task i is given as $T_i = T_i^o + 2 \times T_i^c + T_i^p = s_i/b_{ij} + 2\delta_{jk} + q_i/c_{ik}$.

Thus, the deadline-constrained task mapping $\{x_{ij}, y_{ik}\}$ and resource allocation $\{b_{ij}, c_{ik}\}$ problem that aims to maximize the total system profit can be formulated as follows.

$$(\mathbf{P}_0) \quad \max \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{A}_i} \sum_{k\in\mathcal{N}} x_{ij} y_{ik} p_i \tag{1}$$

subject to:

$$\sum_{j\in\mathcal{A}_i} x_{ij}\frac{s_i}{b_{ij}} + 2\sum_{j\in\mathcal{A}_i}\sum_{k\in\mathcal{N}} x_{ij}y_{ik}\delta_{jk} + \sum_{k\in\mathcal{N}} y_{ik}\frac{q_i}{c_{ik}} \le \Delta_i, \ \ \forall i \in \mathcal{I} \tag{1a}$$

$$\sum_{j\in\mathcal{A}_i} x_{ij} \le 1, \ \ \forall i \in \mathcal{I} \tag{1b}$$

$$\sum_{j\in\mathcal{A}\setminus\mathcal{A}_i} x_{ij} = 0, \ \ \forall i \in \mathcal{I} \tag{1c}$$

$$\sum_{k\in\mathcal{N}} y_{ik} \le 1, \ \ \forall i \in \mathcal{I} \tag{1d}$$

$$\sum_{i\in\mathcal{I}} x_{ij} b_{ij} \le b_j, \ \ \forall j \in \mathcal{A} \tag{1e}$$

$$\sum_{i\in\mathcal{I}} y_{ik} c_{ik} \le c_k, \ \ \forall k \in \mathcal{N} \tag{1f}$$

$$x_{ij}, y_{ik} \in \{0,1\}, \ \forall i \in \mathcal{I}, \ \forall j \in \mathcal{A}, \forall k \in \mathcal{N} \tag{1g}$$

TABLE I
NOTATION (PARAMETERS AND VARIABLES)

Notation	Definition
$\mathcal{I}$	Taskset, where $i \in \mathcal{I}$ denotes a task
$\mathcal{A}$	Access point set, where $j \in \mathcal{A}$ denotes an access point
$\mathcal{A}_i$	Access point subset to which task i can be offloaded
$\mathcal{N}$	Server set, $k \in \mathcal{N}$ denotes a server
s_i	Data size of task i
q_i	Total number of CPU cycles needed by task i
Δ_i	Deadline of task i
p_i	Profit gained by completing task i within deadline
b_j	Bandwidth capacity of access point j
c_k	Computation resource capacity of server k
δ_{jk}	Transmission delay between access point j and server k
b_{ij}	Variable for wireless bandwidth assigned to task i by access point j
c_{ik}	Variable for computation resource assigned to task i by server k
x_{ij}	Binary offloading decision variable, $x_{ij} = 1$ if task i is offloaded to access point j
y_{ik}	Binary processing decision variable, $y_{ik} = 1$ if task i is processed on server k

The constraint (1a) guarantees that the total completion time of a task cannot exceed its deadline. Constraints (1b) and (1c) ensure that a task can only be offloaded to at most one access point in $\mathcal{A}_i$. Constraint (1d) guarantees that a task can only be processed on at most one server. Finally, constraints (1e) and (1f) ensure that the total bandwidth or computation resource assigned to all the tasks by an access point or a server cannot exceed its bandwidth or computation resource capacity. Because of the quadratic terms $x_{ij}y_{ik}, x_{ij}b_{ij}$ and $y_{ik}c_{ik}$ in Eqs. (1), (1a), (1e) and (1f), the nonconvex terms ($x_{ij}\frac{1}{b_{ij}}$ and $y_{ik}\frac{1}{c_{ik}}$) in Eq.(1a), and the binary variables x_{ij} and y_{ik}, problem $\mathbf{P}_0$ is a nonconvex MINLP optimization problem.

III. ZERO-SLACK BASED GREEDY HEURISTIC (ZSG)

The zero-slack based greedy heuristic algorithm (ZSG) for solving problem $\mathbf{P}_0$ comprises three main steps. First, the total available time for each task i (its deadline Δ_i) is distributed into three parts: task offloading time T_i^o, data transmission time through the backhaul network $2\times T_i^c$, and task processing time T_i^p. Given this distribution of total time, we calculate the required bandwidth and computation resource for each task i and for every possible access point-server pair (j, k). Finally, we prioritize all the (i, j, k) options based on a metric, and greedily allocate them whenever feasible. In ZSG, the resources allocated to each provisioned task are just enough for the task to be completed exactly at its deadline. This is possible because any feasible solution of problem $\mathbf{P}_0$ can be converted to a feasible solution that every provisioned task is completed at its deadline without any profit loss.

Deadline Distribution: The deadline of task i is distributed into three parts: T_i^o, $2 \times T_i^c$, and T_i^p. Since the resources are allocated to tasks in the way that every provisioned task is completed exactly at its deadline, $T_i^o + T_i^p = \Delta_i - 2 \times T_i^c$. Suppose γ_{ijk} denotes the fraction of $T_i^o + T_i^p$ used for task i offloading for a given access point-server pair (j, k). That is, $T_i^o = \gamma_{ijk}(\Delta_i - 2 \times T_i^c)$ and $T_i^p = (1 - \gamma_{ijk})(\Delta_i - 2 \times T_i^c)$.

In ZSG, the value of γ_{ijk} is given by,

$$\frac{\gamma_{ijk}}{1-\gamma_{ijk}} = \frac{(\frac{s_i}{\Delta_i})/b_j}{(\frac{q_i}{\Delta_i})/c_k},\ \forall i \in \mathcal{I},\ \forall j \in \mathcal{A}_i,\ \forall k \in \mathcal{N} \quad (2)$$

The main idea for determining γ_{ijk} is that a task with a relatively larger data size will require more time for task offloading, and a task that needs relatively more CPU cycles will require more time for task processing.

Resource allocation calculation: There exist many possible access point-server pairs (j,k) to offload and process a task i. Due to variations in δ_{jk}, the corresponding values for the required bandwidth (b_{ij}) and computation resource (c_{ik}) to meet the task deadline are also different for different (j,k) pair. For an access point-server pair (j,k), suppose b_{ijk} and c_{ijk} denote the bandwidth and computation resource that must be allocated to task i for finishing the task at its deadline. For a given γ_{ijk}, to meet the end-to-end deadline Δ_i of task i, b_{ijk} and c_{ijk} can be calculated as follows.

$$b_{ijk} = \frac{s_i}{\gamma_{ijk}(\Delta_i - 2\delta_{jk})},\ \forall i \in \mathcal{I},\ \forall j \in \mathcal{A}_i,\ \forall k \in \mathcal{N} \quad (3)$$

$$c_{ijk} = \frac{q_i}{(1-\gamma_{ijk})(\Delta_i - 2\delta_{jk})},\ \forall i \in \mathcal{I},\ \forall j \in \mathcal{A}_i,\ \forall k \in \mathcal{N} \quad (4)$$

Algorithm 1 Zero-Slack based Greedy Algorithm (ZSG)

Require: $\mathcal{I}, \mathcal{A}, \mathcal{N}, \delta$
1: $\mathcal{P} \leftarrow \emptyset$;
2: **for** $i \in \mathcal{I}, j \in \mathcal{A}_i, k \in \mathcal{N}$ **do**
3: Calculate $\gamma_{ijk}, b_{ijk}, c_{ijk}$ based on Eqs. (2), (3), and (4);
4: Calculate p_{ijk} based on Eq. (5), $\mathcal{P} \leftarrow \mathcal{P} \cup \{p_{ijk}\}$;
5: **end for**
6: Sort all p_{ijk} values of set $\mathcal{P}$ in non-increasing order;
7: **while** $\mathcal{P} \neq \emptyset$ **do**
8: Determine the (i,j,k) option with the largest $p_{ijk} \in \mathcal{P}$;
9: **if** $b_{ijk} \leq b_j$ and $c_{ijk} \leq c_k$ **then**
10: $x_{ij} \leftarrow 1, y_{ik} \leftarrow 1, b_{ij} \leftarrow b_{ijk}, c_{ik} \leftarrow c_{ijk}$;
11: $b_j \leftarrow b_j - b_{ijk}, c_k \leftarrow c_k - c_{ijk}$;
12: $\mathcal{P} \leftarrow \mathcal{P} \setminus \{p_{i'j'k'} | i' = i, p_{i'j'k'} \in \mathcal{P}\}$;
13: **else**
14: $\mathcal{P} \leftarrow \mathcal{P} \setminus \{p_{ijk}\}$;
15: **end if**
16: **end while**
17: **return** $\mathbf{x}, \mathbf{y}, \mathbf{b}, \mathbf{c}$

Prioritization of tasks and server pairs: Suppose option (i,j,k) denotes the mapping of task i to the access point-server pair (j,k). The priority of option (i,j,k) is denoted as p_{ijk}, and given by the following equation.

$$p_{ijk} = \frac{p_i}{\left(\frac{b_{ijk}}{b_j}\right) \times \left(\frac{c_{ijk}}{c_k}\right)},\ \forall i \in \mathcal{I},\ \forall j \in \mathcal{A}_i,\ \forall k \in \mathcal{N} \quad (5)$$

The intuition behind this metric is that an option (i,j,k) with a higher profit (p_i) and lower resource usage (b_{ijk} and c_{ijk}) should be given higher priority.

The detail steps of ZSG are presented in Algorithm 1. For each (i,j,k) option, calculate γ_{ijk}, b_{ijk} and c_{ijk} based on Eqs. (2), (3) and (4), where $i \in \mathcal{I}, j \in \mathcal{A}_i, k \in \mathcal{N}$ (line 3). Then, calculate p_{ijk} according to Eq. (5), and add p_{ijk} to set $\mathcal{P}$ (line 4). After all possible p_{ijk} values are calculated, sort these p_{ijk} values in nonincreasing order (line 6). If set $\mathcal{P}$ is not empty, the (i,j,k) option with the largest priority value (p_{ijk}) is chosen, and the corresponding mapping and allocation are realized if the resource capacity constraints on access point j and server k are met (lines 7-11). Once a task is provisioned, all the p_{ijk} values related to task i are removed from set $\mathcal{P}$ (line 12). Otherwise, only p_{ijk} is discarded from set $\mathcal{P}$ and the algorithm proceeds with the next (i,j,k) option with largest p_{ijk} value in set $\mathcal{P}$ (line 14). The algorithm stops when set $\mathcal{P}$ is empty.

IV. Linear Discretization Method (LDM)

In this section, we present a linear discretization method (LDM) to solve the nonconvex MINLP problem $\mathbf{P}_0$ by reformulating it to an ILP problem. The LDM assumes that minimum units exist for the allocation of bandwidth and computation resource and any resource allocation will be an integer multiple of corresponding minimum unit. Suppose the minimum unit of bandwidth is denoted as $\tilde{b}$ and that for the computation resource is denoted as $\tilde{c}$. In Problem $\mathbf{P}_0$, we now replace terms as follows: b_{ij} by $u_{ij}\tilde{b}$ and c_{ik} by $v_{ik}\tilde{c}$ where u_{ij} and v_{ik} are nonnegative integer variables, and b_j by $u_j\tilde{b}$ and c_k by $v_k\tilde{c}$ where u_j and v_k are positive integer parameters. Note that u_j and v_k are the upper bounds for u_{ij} and v_{ik}, respectively, for all $i \in \mathcal{I}$.

In the discretized version of problem $\mathbf{P}_0$ (as defined above), the deadline constraint of Eq.(1a) can be rewrite as follows.

$$\sum_{j\in\mathcal{A}_i} x_{ij}\frac{s_i}{u_{ij}\tilde{b}} + 2\sum_{j\in\mathcal{A}_i}\sum_{k\in\mathcal{N}} x_{ij}y_{ik}\delta_{jk} + \sum_{k\in\mathcal{N}} y_{ik}\frac{q_i}{v_{ik}\tilde{c}} \leq \Delta_i,\ \ \forall i \in \mathcal{I} \quad (6)$$

Eq. (6) is still nonconvex because of the terms $x_{ij}\frac{1}{u_{ij}}$ and $y_{ik}\frac{1}{v_{ik}}$. To linearize these terms, the general idea is to discretize one variable and use the summation of finite linear terms to replace the original nonconvex term [9]. Take $x_{ij}\frac{1}{u_{ij}}$ as an example. The variable u_{ij} is mapped into a finite number of possible values; this is feasible because u_{ij} is an integer with a finite range. Each positive u_{ij} value is associated with a new binary variable $x_{ijm} \in \{0,1\}$, $m \in \{1,2,\ldots,u_j\}$, where $\sum_{m=1}^{u_j} x_{ijm} = x_{ij} \leq 1$. The variable x_{ijm} determines which discrete value m is chosen by u_{ij}. $x_{ijm} = 1$ only when the discrete value m is selected and in this case $x_{ij}\frac{1}{u_{ij}} = x_{ijm}\frac{1}{m}$. For the case when $x_{ij} = 0$, $\sum_{m=1}^{u_j} x_{ijm} = 0$ and none of the discrete values is selected. Note, when $u_{ij} = 0$, x_{ij} must be 0 to satisfy the deadline constraint (6) and in this case as well $\sum_{m=1}^{u_j} x_{ijm} = 0$. Thus, the nonconvex term $x_{ij}\frac{1}{u_{ij}}$ can be redefined as follows.

$$x_{ij}\frac{1}{u_{ij}} = \sum_{m=1}^{u_j} x_{ijm}\frac{1}{m},\ \forall i \in \mathcal{I},\ \forall j \in \mathcal{A}_i \quad (7)$$

After the discretization of u_{ij}, we have

$$u_{ij} = \sum_{m=1}^{u_j} x_{ijm} m \leq \sum_{m=1}^{u_j} x_{ijm} u_j \leq x_{ij} u_j. \tag{8}$$

Using the same technique, we can also linearize the term $y_{ik}\frac{1}{v_{ik}}$. Suppose each positive value of v_{ik} is associated with a new binary variable $y_{ikn} \in \{0,1\}$, $n \in \{1,2,\ldots,v_k\}$, where $\sum_{n=1}^{v_k} y_{ikn} = y_{ik} \leq 1$. Then, the nonconvex term $y_{ik}\frac{1}{v_{ik}}$ can be redefined as follows.

$$y_{ik}\frac{1}{v_{ik}} = \sum_{n=1}^{v_k} y_{ikn}\frac{1}{n},\ \forall i \in \mathcal{I},\ \forall k \in \mathcal{N}. \tag{9}$$

$$v_{ik} = \sum_{n=1}^{v_k} y_{ikn} n \leq \sum_{n=1}^{v_k} y_{ikn} v_k \leq y_{ik} v_k. \tag{10}$$

Eqs. (8) and (10) define the property that when task i is not mapped to access point j or server k, where $x_{ij} = 0$ or $y_{ik} = 0$, no corresponding resource will be assigned to task i, which gives $u_{ij} = 0$ or $v_{ik} = 0$. Thus, constraints (1e) and (1f) in the discretized version of Problem $\mathbf{P}_0$ can be rewritten as follows.

$$\sum_{i\in\mathcal{I}} x_{ij} u_{ij} = \sum_{i\in\mathcal{I}} u_{ij} = \sum_{i\in\mathcal{I}} \sum_{m=1}^{u_j} x_{ijm} m \leq u_j,\ \forall j \in \mathcal{A} \tag{11}$$

$$\sum_{i\in\mathcal{I}} y_{ik} v_{ik} = \sum_{i\in\mathcal{I}} v_{ik} = \sum_{i\in\mathcal{I}} \sum_{n=1}^{v_k} y_{ikn} n \leq v_k,\ \forall k \in \mathcal{N} \tag{12}$$

Thus far, we have transformed the nonconvex terms $x_{ij}\frac{1}{u_{ij}}$ and $y_{ik}\frac{1}{v_{ik}}$ into linear terms. In problem $\mathbf{P}_0$, there still exists the quadratic term $x_{ij}y_{ik}$ in the objective function (1) as well as in constraint (6). For the quadratic term $x_{ij}y_{ik}$, we use a new binary variable $z_{ijk} \in \{0,1\}$ to replace it. $z_{ijk} = 1$ only when task i is offloaded to access point j ($x_{ij} = 1$) and processed on server k ($y_{ik} = 1$). Since $\sum_{m=1}^{u_j} x_{ijm} = x_{ij}$ and $\sum_{n=1}^{v_k} y_{ikn} = y_{ik}$, the binary variable z_{ijk} can be defined using the following linear constraints [10].

$$z_{ijk} \geq \sum_{m=1}^{u_j} x_{ijm} + \sum_{n=1}^{v_k} y_{ikn} - 1, \forall i \in \mathcal{I}, \forall j \in \mathcal{A}_i, \forall k \in \mathcal{N} \tag{13}$$

$$z_{ijk} \leq \sum_{m=1}^{u_j} x_{ijm}, \forall i \in \mathcal{I}, \forall j \in \mathcal{A}_i, \forall k \in \mathcal{N} \tag{14}$$

$$z_{ijk} \leq \sum_{n=1}^{v_k} y_{ikn}, \forall i \in \mathcal{I}, \forall j \in \mathcal{A}_i, \forall k \in \mathcal{N} \tag{15}$$

$$z_{ijk} \in \{0,1\}, \forall i \in \mathcal{I}, \forall j \in \mathcal{A}_i, \forall k \in \mathcal{N} \tag{16}$$

Eq.(13) ensures that z_{ijk} is 1 only when both x_{ij} and y_{ik} are 1. Thus, the nonconvex MINLP problem $\mathbf{P}_0$ can be reformulated as an ILP problem, under the assumption that resources are allocated in integer multiples of minimum resource units. Note that in the ILP problem, for given resource capacities (b_j and c_k values), smaller values for $\tilde{b}$ and $\tilde{c}$ will result in increased values for u_j and v_k, thus increasing the number of variables. Although this can improve solution quality, it will also lead to increased runtime.

TABLE II
RANGES USED FOR VARIOUS PARAMETERS

Parameter	Range
$c_k, k \in$ Cloud Servers	80 to 100
$c_k, k \in$ Edge Servers	40 to 60
$b_j, j \in \mathcal{A}$	40 to 100
$\delta_{jk}, j \in \mathcal{A}, k \in \mathcal{N}$	0 to 10
$\lvert\mathcal{A}_i\rvert, i \in \mathcal{I}$	1 to 2
$p_i, i \in \mathcal{I}$	10 to 100
$\Delta_i, i \in \mathcal{I}$	$2\times\delta_{jk}^{\max}+15$ to $2\times\delta_{jk}^{\max}+45$, or $2\times\delta_{jk}^{mean}+15$ to $2\times\delta_{jk}^{mean}+45$

V. EXPERIMENT

In this section, we present the experimental results that evaluate the performances of ZSG and LDM. We generate a variety of synthetic tasksets with different parameter settings and provision them on a fixed edge-cloud architecture. The algorithms are compared in terms of achieved system profit.

A. Taskset Generation

The number of access points and servers in the system are fixed at 20 and 25, respectively. 20 of the 25 servers are edge servers, which are deployed collectively with access points, and the remaining 5 servers are cloud servers. Additionally, all the parameters related to access points and servers, including c_k, b_j and δ_{jk}, are randomly sampled integer values from a pre-defined range in Table II. Note that only the δ_{jk} between the collectively deployed access point j and server k is set to 0. Cloud servers have larger computation resource capacity than that of edge servers. Thus, the capacity range of the cloud servers is from 80 to 100, and the capacity range of the edge servers is from 40 to 60. These values are then kept fixed throughout the experiments[1].

For each task $i \in \mathcal{I}$, its profit p_i, deadline Δ_i, and the number of access points supporting task i offloading ($|\mathcal{A}_i|$) are also randomly sampled integer values from a pre-defined range as shown in Table II. For cost concern, the deployment of access points should avoid too many overlapping coverage areas, thus, the number of access points each task can be offloaded to ranges from 1 to 2. Given $|\mathcal{A}_i|$, the access points to which a task can offload are randomly chosen from the 20 access points. Suppose $\delta_{jk}^{\max} = \max_{j\in\mathcal{A},k\in\mathcal{N}}(\delta_{jk})$ and $\delta_{jk}^{mean} = mean_{j\in\mathcal{A},k\in\mathcal{N}}(\delta_{jk})$. The task deadlines are sampled from two different ranges with equal probability. One of them, $[2 \times \delta_{jk}^{\max} + 15, 2 \times \delta_{jk}^{\max} + 45]$, uses the largest access point to server transmission delay, representing tasks that have relatively more time for offloading and processing. Whereas the other, $[2 \times \delta_{jk}^{mean} + 15, 2 \times \delta_{jk}^{mean} + 45]$, uses the average access point to server transmission delay, and thus represents tasks that have relatively less time for offloading and processing. These task parameters are sampled repeatedly when generating the tasks in each taskset.

[1]Although the edge-cloud architecture parameters are fixed in all our experiments, the taskset parameters are varied across a wide range to evaluate the performance of the algorithms for different resource usage scenarios.

To synthesize tasksets with varying levels of resource usage, we generate tasks with varying bandwidth and computation resource utilizations (amount of resource required in a given time interval). This in effect varies the q_i and s_i values for each task i. For the time interval, we use $\tau_i = \Delta_i - 2 \times \delta_{jk}^{mean}$, which roughly captures the amount of time available to complete both offloading and processing. Thus, for wireless bandwidth, the utilization of a task i offloading to an access point j, ub_{ji}, is defined as $ub_{ji} = s_i/(b_j \times \tau_i)$. Similarly, for computation resource, the utilization of a task i, uc_i, is defined as $uc_i = q_i/(\min_{k \in \mathcal{N}} c_k \times \tau_i)$. For feasibility, we assume ub_{ji} and uc_i are always less than or equal to 1.

To generate tasksets, we consider a different number of tasks in each taskset ($\{40, 60, 80, 100, 120\}$). For the wireless bandwidth, we consider different values for the total bandwidth utilization of each access point ($ub \in \{0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$). Similarly, for the computation resource, we consider different values for the total compute utilization of the entire edge-cloud system ($uc \in \{1, 5, 9, 13, 17, 21, 25, 29, 33, 35.50\}$), where $35.50 = \sum_{k \in \mathcal{N}} c_k / \min_{k \in \mathcal{N}} c_k$ since we normalize uc by $\min_{k \in \mathcal{N}} c_k$. For each combination of these three parameters (400 in all), we generate 30 tasksets, resulting in a total of $12,000$ different tasksets.

To generate a single taskset, given the values for ub, uc and the number of tasks, we first generate the profit p_i, deadline Δ_i and access point set $\mathcal{A}_i$ for each task i as described earlier. Then, for each access point $j \in \mathcal{A}$, given total utilization $ub \times b_j$ and the tasks that can be offloaded to that access point (denoted by set $\mathcal{I}_j$), we use an existing algorithm called Uunifast [11] to generate the task bandwidth utilization values ub_{ji} such that $ub = \sum_{i \in \mathcal{I}_j} ub_{ji}$. This algorithm efficiently generates the task utilization values using uniform random sampling and without any bias. Since a task i can be within the coverage area of more than one access point, it can have different ub_{ji} values assigned to it by this algorithm for each feasible access point j. Therefore, we set its data size s_i as the maximum obtained from those values given by $\max_{j \in \mathcal{A}_i}(ub_{ji} \times b_j \times \tau_i)$. Finally, given total computation resource utilization uc, we use another existing algorithm called Stafford's Randfixedsum [12] to generate the tasks' computation resource utilization values uc_i such that $uc = \sum_{i \in \mathcal{I}} uc_i$. This algorithm uses similar techniques as Uunifast and generates uniformly random and unbiased task utilization values even when the total computation resource utilization uc is greater than 1. We also restrict each uc_i to be no more than 1 to ensure that the compute requirement q_i of every task $i \in \mathcal{I}$ can be satisfied by any server.

For LDM, we consider two different values for the minimum units ($\tilde{c}$ and $\tilde{b}$), 5 and 15, and denote the corresponding LDM as LDM-5 and LDM-15. Once $\tilde{c}$ and $\tilde{b}$ are fixed, v_k and u_j in Eqs. (11) and (12), are set as $\lfloor c_k/\tilde{c} \rfloor$ and $\lfloor b_j/\tilde{b} \rfloor$, respectively. Besides, since the runtimes of the algorithms are generally proportional to the number of tasks in a taskset, we also partition the tasksets based on this number and allocate runtimes to LDM-5 (likewise LDM-15) to be 600 (likewise 200) times the maximum observed runtime for ZSG within each partition. This ensures a fair comparison because LDM-5 has three times more variables than LDM-15 and the ILP solver generally requires orders of magnitude more time than the ZSG heuristic. Experiments were run on a desktop PC with Intel Xeon(R) Gold 5220R 2.2GHz CPU and 128GB of RAM, and Gurobi was used as the ILP solver of LDM[2].

B. Discussion of Results

To visualize the results, we have categorized tasksets based on the resource usage intensity of tasks. A task i is identified to be *computation intensive* if q_i/τ_i is more than 20% of the minimum c_k value. Likewise, a task is identified to be *bandwidth intensive* if s_i/τ_i is more than 20% of the minimum b_j value among all $j \in \mathcal{A}_i$. The *profit gain ratio* is used to compare the performances of different algorithms, which is the ratio of the total profit of tasks provisioned by the algorithm to the total profit of all tasks in the taskset.

$$\text{profit gain ratio} = \frac{\text{total profit of provisioned tasks}}{\text{total profit of all tasks in the taskset}}$$

Results for tasksets with varying levels of computation and bandwidth intensive tasks are shown in Figs. 2 and 3, respectively. In these plots, x-axis denotes the percentage of resource intensive tasks in the taskset, and y-axis denotes the profit gain ratio. For each algorithm the figures show a standard box-plot, with the line (likewise dot) inside the box denoting median (likewise mean). As can be seen from these figures, the performance of ZSG is close to LDM-5 when the percentage of resource intensive tasks is small, but this gap widens as the percentage increases. LDM-5 obtains an average of 2.98% more profit than ZSG for the considered tasksets. This is expected because tasksets with a higher percentage of resource intensive tasks are usually harder to provision, and in these cases, LDM-5 with a standard ILP solver performs better. Note that although LDM-5 outperforms ZSG, their performance gap is small. This is because the

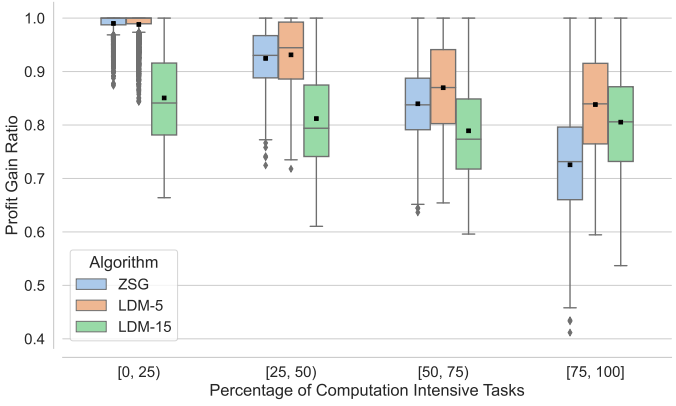

Fig. 2. Performance with varying percentage of computation intensive tasks

[2]Experiments code is available at https://github.com/CPS-research-group/CPS-NTU-Public/tree/GLOBECOM2022.

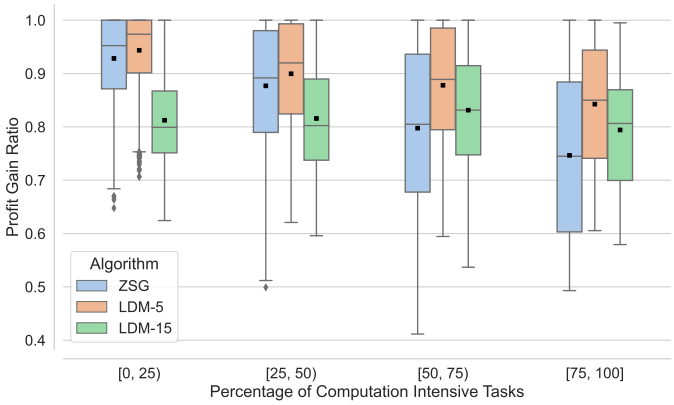

Fig. 3. Performance with varying percentage of bandwidth intensive tasks

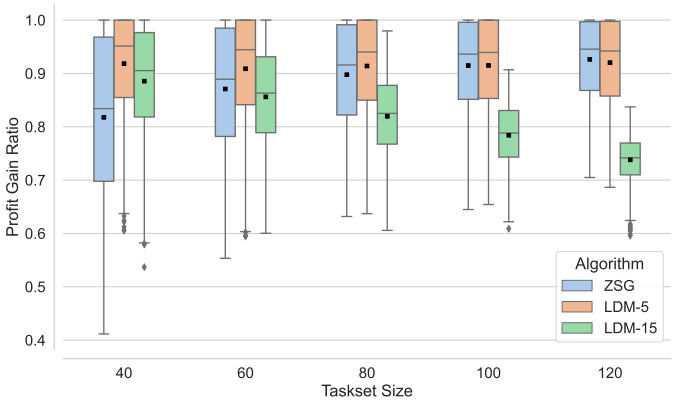

Fig. 4. Performance with varying taskset size

minimum resource units and running time of the ILP solver limit the performance of LDM-5. It can also be observed that the performances of both LDM-5 and ZSG are better than LDM-15. On average, LDM-5 and ZSG obtain 9.87% and 6.88% more system profit than LDM-15, respectively. This shows that the performance of LDM critically depends on the granularity of discretization.

We have also compared the performance of ZSG and LDM with varying taskset sizes, and this result is shown in Fig. 4. Interestingly, the performance of LDM-15 decreases significantly with the increase in taskset size, whereas that of ZSG and LDM-5 reamins the same. As the taskset size increases, the resource requirement of each task generally decreases, thus increasing the resource loss incurred in LDM due to discretization (and hence decreasing achieved profit). The larger base unit in LDM-15 results in a more significant resource loss during task provision and causes LDM-15 to perform worse than LDM-5 and ZSG.

VI. Conclusion

This paper addressed a deadline-constrained multi-resource task mapping and allocation problem for an edge-cloud system. A key challenge of the problem was the allocation of resources on two different units (access point and server) for the same task with an end-to-end deadline. Two effective methods, called ZSG and LDM, were proposed to determine the task mapping and allocation of wireless bandwidth and computation resources. Experimental results demonstrated the efficiency of the proposed methods when dealing with tasksets with a variety of resource requirements. Although LDM with smaller minimum resource units outperformed ZSG, the ILP solver of LDM generally required orders of magnitude more time than ZSG. Thus, ZSG is superior to LDM in large-scale system, and LDM is preferred if the system has hardware acceleration for the ILP solver or requires the solution with a high profit gain ratio.

In this paper, we assume that the data transmission latency is independent of the data size in the backhaul network, which might limit the application of the proposed edge-cloud system in real-world systems. In the future, we would like to explore an edge-cloud system where the data size affects data transmission latency in the backhaul network. Besides, we would also like to explore a distributed and online solution for the presented problem, specifically considering resource scheduling over time.

References

[1] Z. Li, M. A. Uusitalo, H. Shariatmadari, and B. Singh, "5g urllc: Design challenges and system concepts," in *2018 15th International Symposium on Wireless Communication Systems (ISWCS)*, 2018, pp. 1–6.

[2] C. Chekuri and S. Khanna, "A polynomial time approximation scheme for the multiple knapsack problem," *SIAM Journal on Computing*, vol. 35, no. 3, pp. 713–728, 2005.

[3] T. T. Vu, D. N. Nguyen, D. T. Hoang, E. Dutkiewicz, and T. V. Nguyen, "Optimal energy efficiency with delay constraints for multi-layer cooperative fog computing networks," *IEEE Transactions on Communications*, vol. 69, no. 6, pp. 3911–3929, 2021.

[4] Q. Li, J. Zhao, and Y. Gong, "Cooperative computation offloading and resource allocation for mobile edge computing," in *2019 IEEE International Conference on Communications Workshops (ICC Workshops)*, 2019, pp. 1–6.

[5] V. Millnert, J. Eker, and E. Bini, "Achieving predictable and low end-to-end latency for a network of smart services," in *2018 IEEE Global Communications Conference (GLOBECOM)*, 2018, pp. 1–7.

[6] R. Cziva, C. Anagnostopoulos, and D. P. Pezaros, "Dynamic, latency-optimal vnf placement at the network edge," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018, pp. 693–701.

[7] C. Yang, Y. Liu, X. Chen, W. Zhong, and S. Xie, "Efficient mobility-aware task offloading for vehicular edge computing networks," *IEEE Access*, vol. 7, pp. 26 652–26 664, 2019.

[8] S. Ramanathan, N. Shivaraman, S. Suryasekaran, A. Easwaran, E. Borde, and S. Steinhorst, "A survey on time-sensitive resource allocation in the cloud continuum," *it-Information Technology*, vol. 62, no. 5-6, pp. 241–255, 2020.

[9] A. M. Koster and S. Kuhnke, "An adaptive discretization algorithm for the design of water usage and treatment networks," *Optimization and Engineering*, vol. 20, no. 2, pp. 497–542, 2019.

[10] An introduction to mixed integer nonlinear optimization. [Online]. Available: https://www.ima.umn.edu/2015-2016/ND8.1-12.16/25419

[11] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1, pp. 129–154, 2005.

[12] P. Emberson, R. Stafford, and R. I. Davis, "Techniques for the synthesis of multiprocessor tasksets," in *proceedings 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010)*, 2010, pp. 6–11.