QoS-Aware Priority-Based Task Offloading for Deep Learning Services at the Edge

Minoo Hosseinzadeh[1], Andrew Wachal[1], Hana Khamfroush[1], and Daniel E. Lucani[2]

[1]Department of Computer Science, University of Kentucky, USA
[2]Department of Engineering, Aarhus University, Denmark

2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC) | 978-1-6654-3161-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/CCNC49033.2022.9700676

***Abstract*—Emerging Edge Computing (EC) technology has shown promise for many delay-sensitive Deep Learning (DL) based applications of smart cities in terms of improved Quality-of-Service (QoS). EC requires judicious decisions which jointly consider the limited capacity of the edge servers and provided QoS of DL-dependent services. In a smart city environment, tasks may have varying priorities in terms of when and how to serve them; thus, priorities of the tasks have to be considered when making resource management decisions. In this paper, we focus on finding optimal offloading decisions in a three-tier user-edge-cloud architecture while considering different priority classes for the DL-based services and making a trade-off between a task's completion time and the provided accuracy by the DL-based service. We cast the optimization problem as an Integer Linear Program (ILP) where the objective is to maximize a function called gain of system (GoS) defined based on provided QoS and priority of the tasks. We prove the problem is NP-hard. We then propose an efficient offloading algorithm, called PGUS, that is shown to achieve near-optimal results in terms of the provided GoS. Finally, we compare our proposed algorithm, PGUS, with heuristics and a state-of-the-art algorithm, called GUS, using both numerical analysis and real-world implementation. Our results show that PGUS outperforms GUS by a factor of 45% in average in terms of serving the top 25% higher priority classes of the tasks while still keeping the overall percentage of the dropped tasks minimal and the overall gain of system maximized.**

***Index Terms*—Edge Computing, Task Offloading, Resource Management, Deep Learning, Quality-of-Service, Priority**

I. Introduction

The rapid increase of data produced by different devices in smart cities era brings the necessity of developing technologies to process such data. Recently, Deep learning (DL) techniques [1] have emerged as promising technologies to process large scale data. Cloud Computing (CC) has facilitated running different computationally-intensive tasks including DL-dependent services in the past. However, due to long delay caused by the large distance between the cloud server and users, a geographically closer server to the users is needed for delay-sensitive applications such as self-deriving cars [2].

Edge Computing (EC) as a promising paradigm is introduced to solve different issues generated by using CC such as large delay for delay-sensitive applications [3], [4]. Moreover, EC systems have shown a great potential in terms of completion time reduction of serving DL-dependent applications [2], [5]. Although EC has provided many opportunities for different types of applications, edge servers have limited computation, communication, and storage capacities. Therefore, resource management plays a key role in such systems. Different resource management strategies for EC systems such as optimal service placement [2], [6], [7] and optimal task scheduling/offloading [8]–[10] have been introduced in the literature. In this paper we focus on task offloading in the EC systems for running DL-dependent services. DL models are evaluated based on different metrics such as accuracy, Mean Squared Error (MSE), etc., depending on the deep learning task [1]. In this paper, we focus on classification tasks and as such we consider accuracy to evaluate the quality of the provided service. Usually, better DL models have higher accuracy and need more computational resources to run. Both the completion time of serving a task and the accuracy of the DL model used for serving that task play a key role in making the users satisfied in terms of the Quality-of-Service (QoS). Moreover, given the wide range of DL-dependent services offered by smart cities, priority classes have to be defined in order to accommodate different applications based on their importance. For example, serving smart fire detection tasks in a smart home is more important than serving the small plant watering tasks. Additionally, from a service provider's point of view, users should be served based on their payments. As such, considering priority of the tasks is of fundamental importance.

Previous papers that addressed the offloading problem in EC systems can be classified into two categories. The first category focused on maximizing the QoS in EC system [2], [8], [9], [11]–[14] and the second category focused on the priority of the tasks in EC systems [15]–[19]. The first category papers addressed maximizing QoS in EC systems either for a general service type [11]–[14] or specifically for DL-dependent services [2], [8], [9]. These papers focused on different goals [14], such as jointly minimizing both end users' energy consumption and the task's completion time [11], minimizing task completion time [12], reducing latency [13], and maximizing the QoS considering DL-dependent services [2], [8]–[10]. However, none of these papers considered challenges related to *priority* aspects of the tasks. On the other hand, the second category of related work has investigated offloading tasks based on tasks priority in the EC systems [15]–[19], however, to the best of our knowledge none of these works fully investigated the QoS-performance trade-off incurred by serving DL-dependent tasks based on their priorities in a three-tier system. For example, authors in [15]–[19] focused on priority-based offloading decision making from user devices

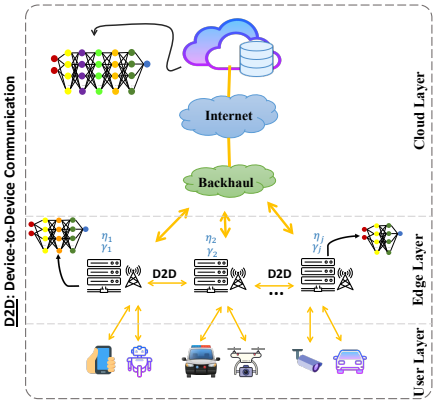

Fig. 1: System Architecture

to only one edge server in a two tier EC system. However, we consider a three tier EC system consisting of multiple edge servers allowed to have Device-to-Device (D2D) communication and offloading the tasks from edge servers to either cloud server or neighbourhood edge server. Additionally, they have not considered the completion time-accuracy trade-off for running DL-dependent services in the EC systems in their proposed offloading decision making approach. The most similar work to this paper is [9]. The main difference between this paper and [9] is we aim at maximizing the *Gain of System* (GoS) as a function of priority of the tasks and the QoS, whereas the focus of [9] is on maximizing the QoS without considering the priority of the tasks. In short, main contributions of this paper are as follow:

- We focus on the problem of optimal offloading decision making while considering tasks' priorities and making optimal trade-off between the completion time of serving DL-based tasks and the provided accuracy of the DL model used for serving that task. We define a new metric, "Gain of System" (GoS), to incorporate all of the above-mentioned criteria.
- We cast the problem as an Integer Linear Programming (ILP) model where the goal is to maximize the GoS. We prove that the problem is NP-hard and propose a greedy algorithm, called PGUS, to solve the proposed ILP in polynomial time.
- We provide numerical analysis and real-world implementation and show PGUS outperforms all of the baseline algorithms in terms of providing optimal trade-off and serving larger numbers of higher priority tasks, thus providing larger GoS.

II. Problem Formulation

System Setup. We consider a three layer cloud-edge-users system as shown in 1. The top layer represents the cloud layer which could be seen as a powerful cloud or a set of cloud servers that are more powerful than the other entities of the system in terms of computation power and communication capability. In this paper we consider one single cloud server, shown by cl, however, the model is designed so that it could be easily generalized to more than one cloud server. At the second layer, a set of edge servers, $\mathcal{M}'$, are located each equipped with a base station. The edge servers are connected to both the cloud server and the neighbourhood edge servers through backhaul network and D2D communication, respectively. Both cloud server and edge servers can serve the tasks in the system. Hence, we define the set of possible servers to serve the tasks as $\mathcal{M} = \{\mathcal{M}' \cup cl\}$. Each server $j \in \mathcal{M}$ has a limited communication capacity η_j and computation capacity γ_j, except the cloud server cl which is assumed to have unlimited communication and computation capacity (i.e. $\eta_{cl} = \infty$ and $\gamma_{cl} = \infty$). For simplicity, hereafter, we assume that the last index of the set $\mathcal{M}$ is always the cloud server. The third layer represents the set of users shown by $\mathcal{N}$ that are connected to the edge layer through a wireless network. We assume that each user $i \in \mathcal{N}$ is directly covered by (connected to) one edge server and users can request tasks by sending their tasks to the edge server covering them. The edge server covering user i is shown by s_i, where we also call it local/source edge server. We assume that a user can only submit one task, and users that submit multiple tasks are modeled by multiple users that are geographically located at the same location. Hence, we will use the term *user* and *task* interchangeably

TABLE I: Model nomenclature and descriptions.

	Notation	Description
Input Data	$\mathcal{N}$	Set of total tasks.
	$\mathcal{M}'$	Set of edge servers.
	$\mathcal{M}$	Set of both the cloud server and the edge servers.
	$\mathcal{K}$	Set of total service types.
	$\mathcal{L}$	Set of total deep learning models.
	$\mathcal{P}$	Set of priority classes.
	cl	The cloud server.
	$A_{p_i}^{min}$	Minimum predefined accuracy that should be provided for task i with priority class p_i.
	a_{ijkl}	Accuracy provided for serving task i by server j using service type k and DL model type l.
	$C_{p_i}^{max}$	Maximum predefined completion time that should be provided for serving task i with priority class p_i.
	c_{ijkl}	Completion time provided for serving task i at server j using service type k and DL model l.
	u_{ijkl}	Communication cost of serving task i on server j for service type k using DL model type l.
	v_{ijkl}	Computation cost of serving task i on server j for service type k using DL model type l.
	s_i	The local edge server that directly covers task i.
	I_{ij}	A matrix showing task i is covered by device j.
	Max_{as}	The best available *accuracy* in the system.
	Min_{as}	The minimum available *accuracy* in the system.
	Max_{cs}	The worst *completion time* in the system.
	Min_{cs}	The minimum *completion time* in the system.
	w_{ai}	The weight of *accuracy* requested by user for the task i.
	w_{ci}	The weight of *completion time* requested by user for the task i.
	T_{ij}^{q}	Average Queue delay of task i at edge server j.
	T_{ijkl}^{comp}	Average processing time of task i at server j for service type k using DL model type l.
	$T_{ijj'}^{comm}$	Average communication time needed to send task i from server j to server j'.
	γ_j	Computation capacity of device j.
	η_j	Communication capacity of device j.
	$\omega, \omega', \omega''$	Three real values for GoS function definition.
D. V	X_{ijkl}	A decision variable to show whether the task i will be served at server j for service type k using DL model type l or not.

throughout the paper. We consider a time-slotted system where at a given time slot there are $\mathcal{N}$ tasks in the system and $\mathcal{M}$ possible edge/cloud servers to serve the tasks. We also define $\mathcal{P}$ as the set of different priority classes indexed by p where each task $i \in \mathcal{N}$ belongs to the priority class p_i. Each priority class p has a minimum preferred accuracy and maximum preferred completion time. Users can have tasks with different priority classes. We assume that system has provided $\mathcal{K}$ different service types indexed by k where each service type can be served by several different DL models $\mathcal{L}$ indexed by l each providing different levels of accuracy and consuming different levels of computation resources. Our goal is to find optimal offloading decisions while maximizing the *Gain of System (GoS)* which is defined as a function of *User Satisfaction (US)* and the priority class of each served task.

GoS Definition. We define the GoS for serving task i at server j using service type k and DL model l as:

$$GoS_{ijkl} = \begin{cases} \omega^{(\hat{p_i} \times US_{ijkl})} & \text{if } \Delta a_{ijkl} \geq 0 \wedge \Delta c_{ijkl} \geq 0 \\ \omega'^{(\hat{p_i} \times US_{ijkl})} & \text{if } \Delta a_{ijkl} < 0 \oplus \Delta c_{ijkl} < 0 \\ \omega''^{(\hat{p_i} \times US_{ijkl})} & \text{if } \Delta a_{ijkl} < 0 \wedge \Delta c_{ijkl} < 0 \\ 0 & \textit{dropping the task} \end{cases} \tag{1}$$

where ω, ω', and ω'' are hyper-parameters such that $\omega'' \leq \omega' \leq \omega$. In Eq. 1, $\wedge$ is the *and* operator and $\oplus$ is the *exclusive or* operator. We define $0 < \hat{p_i} \leq 1$ as $\hat{p_i} = \frac{p_i}{p_{max}}$ that represents the normalized value of priority class p_i where p_{max} is the highest value of priority classes values. Note that the higher p value represents the higher priority class (i.e. the more important classes have higher p value). US_{ijkl} represents the US for serving task i at server j for service type k using DL model type l which is defined in Eq. 2. Δc_{ijkl} and Δa_{ijkl} represent difference between the level of service (in terms of completion time and accuracy, respectively) that the user expected to receive and the level of service that is actually provided by the system, and are defined in Eq. 4 and Eq. 3.

US Definition. We define the user satisfaction, US_{ijkl}, for serving task i at server j for the requested service type k using deep learning model l based on:

$$US_{ijkl} = w_{ai}(\Delta a_{ijkl}) + w_{ci}(\Delta c_{ijkl}) \tag{2}$$

where w_{ai}, and w_{ci} are weights (hyper-parameters) assigned to each of the US criteria–accuracy and completion time, respectively. We define Δa_{ijkl} and Δc_{ijkl} as the difference between the level of service that user expected to receive and the level of service that we provide as following:

$$\Delta a_{ijkl} = \frac{a_{ijkl} - Min_{as}}{Max_{as} - Min_{as}} - A_{p_i}^{min} \tag{3}$$

where $0 \leq A_{p_i}^{min} \leq 1$ represents the normalized minimum preferred accuracy for priority class p of task i. a_{ijkl} is the provided accuracy for the tasks i by server j using service type k and DL model l. Max_{as} and Min_{as} are defined as maximum possible accuracy and minimum possible accuracy in the system, respectively. Similarly, we define the Δc_{ijkl} as the difference between provided completion time and the completion time that the user expects to receive as following:

$$\Delta c_{ijkl} = C_{p_i}^{max} - \frac{c_{ijkl} - Min_{cs}}{Max_{cs} - Min_{cs}} \tag{4}$$

where $0 \leq C_{p_i}^{max} \leq 1$ shows the normalized maximum preferred completion time for task i belonging to priority class p. c_{ijkl} is the provided completion time for the tasks i by server j using service type k and DL model l. Max_{cs} and Min_{cs} are respectively the worst possible completion time and the best possible completion time in the system.

The task will be counted as *served-satisfied*, iff the provided accuracy and the provided completion time are equal or better than the requirements of that task's priority class. Otherwise, they might receive less than their priority class requirements, which we call *served-not satisfied*. In the case that there is no capacity remaining in the system, the tasks will be *dropped* and so the GoS for that task will be zero. Now, we define the completion time of serving a task, c_{ijkl}.

Completion time. We assume that each task may experience the following types of delay throughout the system: *queuing delay*, *processing delay*, and *communication delay* (due to offloading). We ignore the communication delay due to sending the tasks (and their dependencies) from users to the edge servers and receiving the results from the edge servers, as this delay is equal for any decision made (e.g., offloading vs. local processing) and the focus of this work is on finding the optimal task offloading decision at the edge servers considering the priority of the tasks. Moreover, we suppose that the delay due to running the decision making algorithms at each edge server is negligible. The running time of our proposed decision making algorithm is discussed in §III. Now, we formally define each type of delay that we consider in this paper: ***1) Queuing Delay*** (T_{ij}^{q})**.** We consider admission control queuing delay (queuing delay due to accepting tasks into the system) and assume that queuing delay due to processing is negligible. Hence, we assume the queuing delay equal to the time between the arrival of task i to edge server j and when a decision regarding how to process the task is made on that edge server. We consider a time-slotted scenario where at each time slot, the tasks that arrived at edge server j will be held in a queue until a decision is made at the end of a time frame. Each time frame consists of multiple time slots. We assume that each edge server broadcasts its remaining communication and computation capacity and average computation, communication, and queuing delay to the neighbourhood edge servers frequently, so, we assume these values are known. ***2) Computation Delay*** (T_{ijkl}^{comp})**.** We consider the computation delay to be the time that takes for the task i to be processed on server $j \in \mathcal{M}$ using service type k and deep learning model type l. ***3) Communication Delay*** ($T_{ijj'}^{comm}$)**.** Communication delay is assumed to be the the time needed to send task i from edge server j to another edge/cloud server $j' \in \mathcal{M}$ in the case that an offloading decision is made. Now, we formally define the completion time of serving a task i at the j-th server using service type k and DL model type l as:

$$c_{ijkl} = \begin{cases} T^{q}_{is_i} + T^{comp}_{ijkl} & \text{if } j = s_i \\ T^{comm}_{is_ij} + T^{q}_{is_i} + T^{q}_{ij} + T^{comp}_{ijkl} & \text{if } j \neq s_i \end{cases}$$

where s_i is the local server covering the task (user) i.

Model Definition. Now, we formally define the ILP model where the goal is to maximize the gain of the system as:

$$\text{Max:} \sum_{i\in\mathcal{N}, j\in\mathcal{M}} \sum_{k\in\mathcal{K}, l\in\mathcal{L}} (GoS_{ijkl}\ X_{ijkl}) \tag{5}$$

$$\text{s.t.:} \sum_{j\in\mathcal{M}, k\in\mathcal{K}, l\in\mathcal{L}} X_{ijkl} \leq 1, \qquad \forall i \in \mathcal{N}, \tag{5a}$$

$$\sum_{i\in\mathcal{N}, k\in\mathcal{K}, l\in\mathcal{L}} X_{ijkl}\ v_{ijkl} \leq \gamma_j, \qquad \forall j \in \mathcal{M}, \tag{5b}$$

$$\sum_{j'\in\mathcal{M}\neq j} \sum_{i\in\mathcal{N}, k\in\mathcal{K}, l\in\mathcal{L}} I_{ij}\ X_{ij'kl}\ u_{ijkl} \leq \eta_j, \quad \forall j \in \mathcal{M}, \tag{5c}$$

$$X_{ijkl} \in \{0, 1\}, \qquad \forall i \in \mathcal{N}, j \in \mathcal{M}, k \in \mathcal{K}, l \in \mathcal{L} \tag{5d}$$

where $\mathbf{X} \triangleq (X_{ijkl})\ \forall i \in \mathcal{N}, j \in \mathcal{M}, k \in \mathcal{K}, l \in \mathcal{L}$ is the set of decision variables and $X_{ijkl} = 1$ iff task i is served by device j using service k and DL model l, and 0 otherwise. The computation cost and the communication cost of serving task i on server j for service type k using model type l are represented by v_{ijkl} and u_{ijkl}, respectively. The computation capacity and the communication capacity of server j are shown by γ_j and η_j, respectively. I is an $|N| \times |M|$ matrix where $I_{ij} = 1\ (\forall i \in \mathcal{N}, j \in \mathcal{M})$ if and only if (user) task i is directly covered by edge device j, and 0 otherwise (i.e. $I_{ij} = 1$ where $j = s_i$). Constraint (5a) ensures that each task will be served by only one server using one service and one DL model or it will be dropped. Constraint (5b) guarantees that the total computation cost needed to process all tasks served by device j should be less than or equal to the computation capacity of device j. Note that we assume cloud server has unlimited computation capacity, hence $\gamma_{cl} = \infty$. Constraint (5c) ensures that the total communication cost required to offload the tasks from device j to other servers should be less than or equal to the overall communication capacity of device j. Constraint (5d) defines the binary decision variables. Depending on the value of decision variable X_{ijkl}, one of the following occurs: 1) *Local processing.* The task will be served on the edge server if $X_{ijkl} = 1$ & $j = s_i$; 2) *Offloading.* The task will be offloaded to either a cloud server or one of the neighboring edge servers if $X_{ijkl} = 1$ & $j \neq s_i$; 3) *Drop.* The task will not be served and will be dropped if $X_{ijkl} = 0$. Note that in 1) and 2) cases, the task could be either *served-satisfied* or *served-not satisfied*.

Theorem 1. *The proposed optimization problem is NP-hard.*

Proof. We prove Theorem 1 through a reduction from the MUS problem initially studied in our previous paper [9]. The MUS problem is proved to be NP-hard using a reduction from the NP-hard Maximum Cardinality Bin Packing (MBCP) problem. By setting $p_i = 1\ (\forall i \in N)$ in Eq. (2), then MUS would be a special case of this problem (i.e. this problem is the general problem). The proposed ILP is then equivalent to the MUS problem and can be solved with a solution to the MUS problem. This concludes the proof. □

Algorithm 1: Proposed Priority-Greedy Algorithm (PGUS)

Input : Given $\mathcal{N}$, $\mathcal{M}$, $\mathcal{K}$, $\mathcal{L}$, $\mathcal{P}$,I, Max_{as}, Max_{cs}, Min_{as}, Min_{cs}, A, C

Output: Choosing X_{ijkl} for each task i

1 $\mathcal{N} \leftarrow$ sort $\mathcal{N}$ based on higher $p \in \mathcal{P}$;
2 **foreach** *task* $i \in \mathcal{N}$ **do**
3 $s_i \leftarrow \{j \mid I_{ij} = 1\}$;
4 $k \leftarrow$ requested service by user i;
5 Initialize G as a list of tuples;
6 **foreach** *server* j *in* $\mathcal{M}$ *which has service type* k **do**
7 **foreach** *model* l *placed on* j *for service type* k **do**
8 $X_{ijkl} \leftarrow 0$;
9 Calculate GoS using Eq. (1);
10 *Append* (Gos, j, l) to G;
11 $sortedGoS \leftarrow$ sort G based on first element (GoS value) in descending order;
12 **foreach** *server* $j \in sortedGoS$ **do**
13 **if** $j = s_i$ **then**
14 **if** $v_{ijkl} \leq \gamma_j$ **then**
15 $X_{ijkl} \leftarrow 1$; `// Locally process task` i `on edge server` $j = s_i$
16 update γ_j;
17 break;
18 **else**
19 **if** $u_{ijkl} \leq \eta_{s_i}$ *and* $v_{ijkl} \leq \mathbb{E}[\gamma_j]$ **then**
20 $X_{ijkl} \leftarrow 1$; `// Offload task` i `to server` j
21 update γ_j and η_j;
22 break;
23 **if** $X_{ijkl} = 0\ \forall j \in \mathcal{M}$ **then**
24 Drop i;

III. Proposed Algorithm

Given that the proposed ILP is NP-hard based on Theorem 1, we propose a greedy algorithm to solve the problem in polynomial time (see Alg. 1). The idea of the algorithm is simple: First, all of the tasks are sorted based on their priority classes in a descending order (higher priority classes go first) (line 1). Then, for each task the algorithm finds the local edge server (s_i) of it (lines 2,3) and its requested service, k (line 4). We select set of all candidate servers which have at least one already placed DL model l for service type k (line 6,7). Next, we calculate the completion time of serving task i using the selected server sets and their relevant service type and DL model types, i.e., $c_{ijkl}\ \forall j \in \mathcal{M}, k \in \mathcal{K}, l \in \mathcal{L}$ (line 9). We assume that a_{ijkl} is known based on the accuracy of the DL model type l available on server j for service type

k. Then we calculate the expected GoS based on c_{ijkl} and a_{ijkl} values (line 10), i.e., $GoS_{ijkl}\ \forall j \in \mathcal{M}, k \in \mathcal{K}, l \in \mathcal{L}$. We sort the candidate servers j based on the value of GoS, descendingly (line 11), and then start from the top server with the highest GoS (line 12). If the selected server is the local server (s_i), the candidate edge server must have enough computation capacity (line 13-17). If the task is offloaded, both the communication capacity of the local server and the computation capacity of the serving place must be available (line 18-22). If the algorithm does not find any server with enough capacity, the task will be dropped (line 23-25). The process will be repeated until there is no task remaining in the queue.

Algorithm Complexity. The complexity of the proposed PGUS algorithm in the worst case is of order $O((|\mathcal{N}|\log|\mathcal{N}|) + |\mathcal{N}|(|\mathcal{L}||\mathcal{M}| + |\mathcal{M}||\mathcal{L}|(\log|\mathcal{M}||\mathcal{L}|) + |\mathcal{L}||\mathcal{M}|)) = O(|\mathcal{N}||\mathcal{L}|^2|\mathcal{M}|^2)$. This is because, sorting tasks in line 1 is of order size of tasks, $|\mathcal{N}|\log|\mathcal{N}|$. Sorting the candidate combination of servers and models at line 10 at most is of order $|\mathcal{N}|(|\mathcal{M}||\mathcal{L}|(\log|\mathcal{M}||\mathcal{L}|))$ and checking each of these combinations at most is of order $|\mathcal{N}|(|\mathcal{M}||\mathcal{L}|)$.

PGUS In Real-World Implementation. For the sake of real-world implementation, we need to approximate average delays of the system to calculate c_{ijkl} given that the completion time of a task may not be fixed and will change over time. We use a `D2D` sub-procedure where edge servers announce their remaining resources and their average queue and average computation delay and average communication speed to other edge servers every one second. We use three other sub-procedures to estimate each type of delay. `PredictCommDelay` predicts the expected communication delay, $\mathbb{E}_{sw}[T^{comm}_{is_ij}]$, for each task i based on size(i)/$\mathbb{E}_{sw}$[speed] where size(i) is the data size of task i and $\mathbb{E}_{sw}$[speed] is the expected transmission speed in the sliding window of the size sw based on the most recent update received from the `D2D` sub-procedure. Similarly, `PredictCompDelay` calculates the expected computation delay, $\mathbb{E}_{sw}[T^{comp}_{ijkl}]$, based on historical data using information obtained from the `D2D` sub-procedure. For the queue delay, `PredictQueueDelay` uses the time that a task joins the queue until the time a decision is made for the local processing case ($T^q_{is_i}$); and $\mathbb{E}_{sw}[T^q_{ij}] = T^q_{is_i} + rand(\frac{\mathbb{E}_{sw}[T^q_{ij'}]}{2}, \mathbb{E}_{sw}[T^q_{ij'}])$ for the offloading case where $\mathbb{E}_{sw}[T^q_{ij'}]$ is the most recent update of average queue delay received from server j'.

IV. Results

System Setup For Numerical Analysis. For the numerical results, we use the following system setup, unless stated otherwise. A set up consisting of nine edge servers and one cloud server has been used. The edge servers are categorized based on their storage, communication, and computation capacity into three categories. The communication capacity of the network is equal to 600 bytes/milliseconds (ms) on average. The number of servers, services, and DL models providing each service is set to $|\mathcal{M}| = 10$, $|\mathcal{K}| = 50$, $|\mathcal{L}| = 10$, respectively. We set $\omega = 64$, $\omega' = 4$, and $\omega'' = 2$. We assume that services are randomly placed on the edge servers based on their associated storage capacity. We define 20 different priority classes (i.e. $|\mathcal{P}| = 20$) where the higher value of p represents the higher priority. Different combinations of completion time and accuracy are defined for priority classes. The preferred completion time and accuracy for each priority class $p \in \mathcal{P}$ is randomly selected between the range of 8000 ms and 17500 ms and 45% and 80%, respectively, such that the higher priority classes are more delay-sensitive and require higher accuracy. The maximum completion time in the system, Max_{cs}, is 25000 ms and the minimum of that, Min_{cs}, is 1000 ms. The maximum provided accuracy in the system, Max_{as}, is 100% and the minimum of that, Min_{as}, is 41%. We set $w_{ai} = w_{ci} = 0.5$, hence, both the Δa_{ijkl} and the Δc_{ijkl} have equal weights. Additionally, for the numerical results, we assume that all the hyper-parameters of the system needed to calculate the GoS such as c_{ijkl} are given/known. For the real-world implementation these need to be measured by sending some control messages as explained in §III.

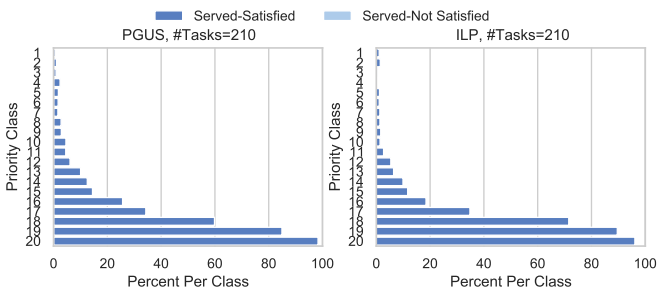

Fig. 2: Numerical results: PGUS vs. ILP

ILP vs PGUS. Given the NP-hardness of the proposed problem, solving the ILP and finding the optimal solution is infeasible for large system setups. So, we evaluate the performance of the PGUS with respect to the solution obtained by the ILP solver for small system setups. We use Python for implementing our PGUS algorithm and Python Pulp package to implement the ILP. A set-up consisting of nine edge servers and one cloud server is used. We run the tests for a small number of tasks $|\mathcal{N}| = 210$ and run each test over 100 Monte Carlo runs. We explore the percent of served-satisfied and served-not satisfied of ILP solver vs. that of PGUS. Based on Fig. 2, PGUS closely matches the results of the ILP solver in terms of serving tasks from the higher priority classes. Additionally, total GoS for PGUS is almost 80% of that of the ILP solver. We test a similar scenario with different values of $\omega = 4$, $\omega' = 2$, and $\omega'' = 1$. The total GoS for PGUS is almost 81% of that of the ILP solver, so different values of ω, ω' and ω'' do not decrease the performance of PGUS. Now, we focus on numerical results for a larger number of tasks.

Baseline Algorithms. We implement different baseline algorithms to evaluate the performance of PGUS. The baseline algorithms are listed as following: ***1) GUS.*** GUS is based on [9] where a greedy algorithm is proposed for offloading decision making whilst making a trade-off between the accuracy and the completion time for DL-dependent services. GUS does not distinguish between the priority of the tasks. Moreover, GUS assumes hard constraints regarding the provided accuracy

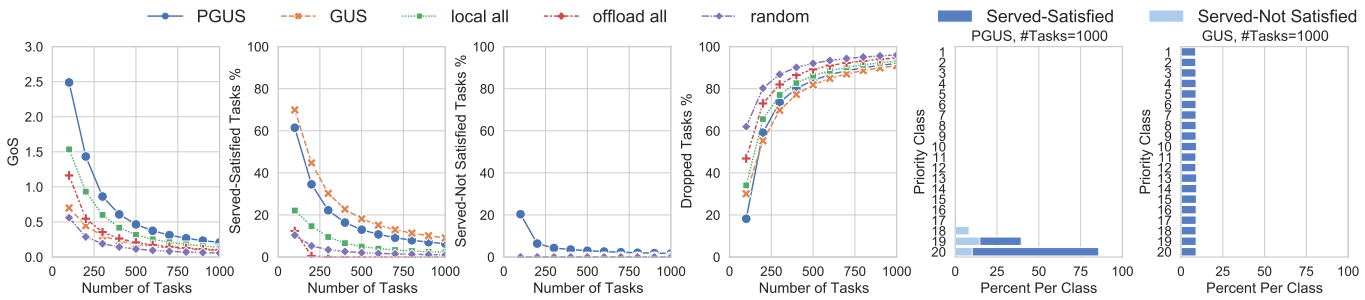

Fig. 3: Numerical results: PGUS vs. baseline algorithms performance comparison

and the provided completion time; i.e., the provided accuracy and provided completion time should be equal or better than what the user requested. ***2) Random.*** This baseline sorts the tasks based on their priority at first. Then, one of the servers (edge/cloud) is randomly chosen to serve the tasks. If the server has enough computation capacity and the source edge server, s_i, also has enough communication capacity to offload the task, the server will be selected. Otherwise, the task will be dropped. ***3) Offload All.*** This baseline offloads all the tasks from each edge server to the cloud, after sorting them based on the priority. Therefore, it sequentially servers the tasks with highest priority until there is no remaining communication capacity to offload the tasks. ***4) Local All.*** This baseline first sorts the tasks based on their priority, and then tasks will be processed on their source/local edge server sequentially until there will be no remaining local computation resources.

Numerical Results. The numerical analysis are provided using MATLAB 2019b. Each data point is an average of 20000 Monte Carlo runs. We change the number of tasks, $\mathcal{N}$, from 100 to 1000 and compare the performance of PGUS with that of all provided baselines in Fig. 3 in terms of GoS, served-satisfied tasks percent, served-not satisfied tasks percent, and dropped tasks percent. Although the percent of served-satisfied tasks of GUS is higher than that of PGUS, the total percent of served tasks of PGUS is larger than that of GUS (less number of dropped tasks in PGUS vs. GUS). The percent of served-not satisfied of GUS is zero as it drops the tasks which it cannot satisfy. As the GUS algorithm provides surpassing results compared to other baseline algorithms in terms of dropped percent of tasks, we focus on comparing PGUS and GUS in terms of percent of served-satisfied and served-not satisfied based on each priority class. Based on Fig. 3 (5th and 6th columns), PGUS serves three higher priority classes by a factor of almost 60% more than that of GUS on average (note that both dark blue bars and light blue bars start from zero) while still providing a smaller percentage of dropped tasks.

Testbed Setup. We assess our proposed scheme using a real-world testbed which consists of a cloud server, two edge servers, and three user devices. A Linux desktop (Intel core i5-3.20 GHz, 8GB RAM) serves as the cloud server. The cloud server is connected to a roughly 8 meters away NETGEAR R6020 router through the wireless connection. The two edge servers are connected to the router through the wireless connection. We provide a heterogeneous set of edge devices with one NVIDIA Jetson Nano Kit (JN) (Quad-core, 4GB RAM) and one Raspberry Pi (RP)4B (Quad core 64-bit, 4GB RAM). The JN is roughly 4 meters from the router. The RP4B is roughly 10 meters from the router. In addition to being connected to the cloud server through the router, the edge servers are connected to each other through a wireless ad-hoc connection as D2D connection. The edge servers are roughly 14 meters apart. We then have three RP3B that are connected to the edge servers as the users through wireless ad-hoc connections. Two RP3B are connected to the JN, and one RP3B is connected to the RP4B. Each user is roughly 20 cm away from its respective edge server device. The RP4B and its user are separated from the rest of the devices by two walls to add some interference in wireless communication.

Testbed Implementation. We consider two pre-trained DL models to provide image recognition service: *GoogLeNet* [20] and *SqueezeNet* [21]. GoogLeNet is placed exclusively on the cloud server, while SqueezeNet is used on the edge servers due to its lighter resource consumption (at the cost of lower accuracy). The images that are used in each service task are from the ImageNet dataset [22]. We provide a client-server program written in C++ to manage the communication between the entities in the system. We have three users where each user creates a task every 500 ms, assigns a random priority class number, p_i, between 1 and 20 (where 20 represents the top priority class) to it, and sends the task to the connected edge server. Each server creates 1000 task in total (i.e. $|\mathcal{N}| = 3000$). The priority class corresponds to a preferred accuracy, which ranges from 43% to 89%, and a normalized preferred completion time, ranging from 0.4 to 1.0. The simulated bandwidth between the RP4B and the cloud server is 7 bytes/ms, and between the JN and the cloud server is 9 bytes/ms. We use this system setup for all implemented algorithms. The edge servers run the algorithm when either the queue is full or a time limit is passed. Additionally, we limit the computation and communication capacity, γ and η, of each edge server to 15 process and 10 tasks for JN and 10 process and 10 tasks for RP4B, respectively, and the queue size of the JN and RP4B to 15 and 10, respectively. The edge servers also share a control message, `D2D` sub-procedures, with each other every 1 sec. The cloud server simply accepts all tasks that are offloaded to it and processes them. We simulate large

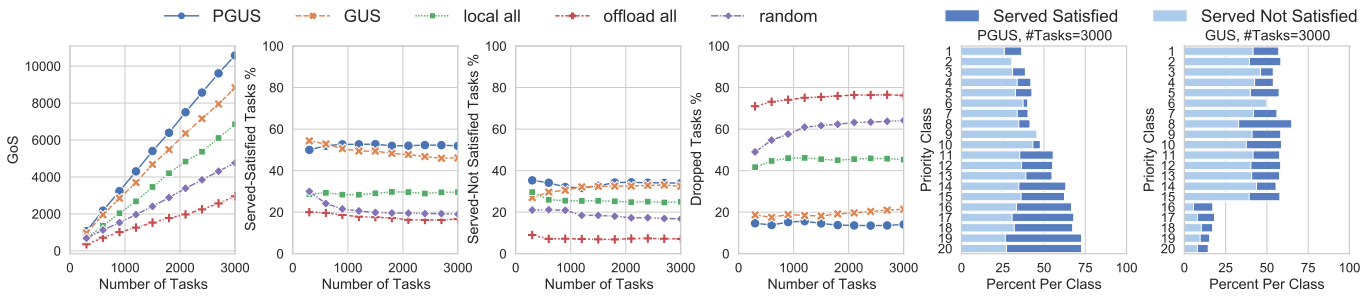

Fig. 4: Testbed implementation: PGUS and baseline algorithms performance comparison

delay between the edge servers and the cloud server due to the long distances by enforcing a synthetic sleep mode for the cloud server before responding to the edge servers tasks.

Testbed Results. Fig. 4 represents the GoS, percentage of served-satisfied tasks, percentage of served-not satisfied tasks, percentage of dropped tasks for PGUS and all baseline algorithms, and percentage of served-satisfied vs. served-not satisfied tasks based on each priority class for the PGUS and GUS algorithms. On average, our results show that PGUS provides more than 9% higher GoS compared to GUS. Fig. 4 implies that not only PGUS satisfies more tasks of higher priority classes compared to the best baseline algorithm, GUS, but it also drops fewer tasks compared to all of the baselines.

V. Conclusion

We propose a new offloading scheme which considers the priority of the tasks in a three-tier users-edge-cloud system for DL-dependent services. We present the optimal offloading decision problem as an ILP model where the goal is to maximize the gain of system defined based on an accuracy-time trade-off of serving the DL-dependent services considering the limited capacity of the edge servers. We prove the proposed problem is NP-hard and propose an offloading scheme which provides near-optimal solutions compared to ILP model. Upon vetting our offloading decision algorithm using both numerical results and real-world implementation, we show that our proposed algorithm can serve the top 25% higher priority classes of tasks by a factor of 45% more than the state-of-the-art algorithm while still keeping the overall percentage of the dropped tasks minimal and the overall gain of system maximized.

Acknowledgements

This work is funded by research grants provided by the National Science Foundation (NSF) and the Cisco Systems Inc. under the grant numbers 1948387 and 1215519250, respectively.

References

[1] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. P. Reyes, M.-L. Shyu, S.-C. Chen, and S. Iyengar, "A survey on deep learning: Algorithms, techniques, and applications," *ACM Computing Surveys (CSUR)*, 2018.

[2] N. Hudson, H. Khamfroush, and D. E. Lucani, "Qos-aware placement of deep learning services on the edge with multiple service implementations," *2021 IEEE ICCCN*, pp. 1–8, 2021.

[3] ETSI, "Mobile edge computing - introductory technical white paper," Sept. 2014.

[4] A. Adeniran, M. A. Hasnat, M. Hosseinzadeh, H. Khamfroush, and M. Rahnamay-Naeini, "Edge layer design and optimization for smart grids," in *2020 IEEE SmartGridComm*, pp. 1–6, IEEE, 2020.

[5] N. Hudon, J. Hossain, M. Hosseinzadeh, H. Khamfroush, M. Rahnamay-Naeini, and N. Ghani, "A framework for edge intelligent smart distribution grids via federated learning," in *2021 ICCCN*, IEEE, 2021.

[6] T. He, N. Bartolini, H. Khamfroush, I. Kim, L. Ma, and T. La Porta, "Service placement for detecting and localizing failures using end-to-end observations," in *IEEE 36th ICDCS*, 2016.

[7] V. Farhadi, F. Mehmeti, T. He, T. F. L. Porta, H. Khamfroush, S. Wang, K. S. Chan, and K. Poularakis, "Service placement and request scheduling for data-intensive applications in edge clouds," *IEEE/ACM Transactions on Networking*, pp. 1–14, 2021.

[8] X. Zhao, M. Hosseinzadeh, N. Hudson, H. Khamfroush, and D. E. Lucani, "Improving accuracy-latency trade-off of edge-cloud computation offloading for deep learning services," in *IEEE Globecom Workshop on Edge Learning over 5G Networks and Beyond*, 2020.

[9] M. Hosseinzadeh, A. Wachal, H. Khamfroush, and D. E. Lucani, "Optimal accuracy-time trade-off for deep learning services in edge computing systems," in *IEEE ICC*, 2021.

[10] M. Hosseinzadeh, N. Hudson, X. Zhao, H. Khamfroush, and D. E. Lucani, "Joint compression and offloading decisions for deep learning services in 3-tier edge systems," in *IEEE DySPAN*, 2021.

[11] M. Kamoun, W. Labidi, and M. Sarkiss, "Joint resource allocation and offloading strategies in cloud enabled cellular networks," in *IEEE ICC, 2015*.

[12] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *IEEE ISIT*, 2016.

[13] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, 2020.

[14] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, 2017.

[15] L. Li, M. Siew, Z. Chen, and T. Q. Quek, "Optimal pricing for job offloading in the mec system with two priority classes," *IEEE Trans. on Vehicular Technology*, 2021.

[16] L. Gao and M. Moh, "Joint computation offloading and prioritized scheduling in mobile edge computing," in *2018 International Conference HPCS*, pp. 1000–1007, IEEE, 2018.

[17] P. Paymard and N. Mokari, "Resource allocation in pd-noma–based mobile edge computing system: Multiuser and multitask priority," *Transactions on Emerging Telecommunications Technologies*, p. e3631, 2019.

[18] S. Guo, D. Wu, H. Zhang, and D. Yuan, "Resource modeling and scheduling for mobile edge computing: A service provider's perspective," *IEEE Access*, vol. 6, pp. 35611–35623, 2018.

[19] Z. Zhu, J. Peng, X. Gu, H. Li, K. Liu, Z. Zhou, and W. Liu, "Fair resource allocation for system throughput maximization in mobile edge computing," *IEEE Access*, vol. 6, pp. 5332–5340, 2018.

[20] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *IEEE conf. on computer vision and pattern recognition*, 2015.

[21] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint: 1602.07360*, 2016.

[22] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.