Optimal Accuracy-Time Trade-off for Deep Learning Services in Edge Computing Systems

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

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

ICC 2021 - IEEE International Conference on Communications | 978-1-7281-7122-7/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICC42927.2021.9500744

***Abstract*—With the increasing demand for computationally intensive services like deep learning tasks, emerging distributed computing platforms such as edge computing (EC) systems are becoming more popular. Edge computing systems have shown promising results in terms of latency reduction compared to the traditional cloud systems. However, their limited processing capacity imposes a trade-off between the potential latency reduction and the achieved accuracy in computationally-intensive services such as deep learning-based services. In this paper, we focus on finding the optimal accuracy-time trade-off for running deep learning services in a three-tier EC platform where several deep learning models with different accuracy levels are available. Specifically, we cast the problem as an Integer Linear Program, where optimal task scheduling decisions are made to maximize overall user satisfaction in terms of accuracy-time trade-off. We prove that our problem is NP-hard and then provide a polynomial constant-time greedy algorithm, called GUS, that is shown to attain near-optimal results. Finally, upon vetting our algorithmic solution through numerical experiments and comparison with a set of heuristics, we deploy it on a test-bed implemented to measure for real-world results. The results of both numerical analysis and real-world implementation show that GUS can outperform the baseline heuristics in terms of the average percentage of satisfied users by a factor of at least 50%.**

***Index Terms*—Mobile edge computing, task offloading, resource management, deep learning, raspberry pi, user satisfaction, quality of experience.**

I. Introduction

The promise of *edge computing* (EC) [1] has sparked ever-increasing attention in recent years. Much of this attention is a result of ever-increasing consumption and generation of data at the network edge in IoT systems. Because conventional cloud computing relies on a cluster of remote hardware resources, this invites issues for delay-sensitive applications that are becoming more prevalent. For instance, self-driving cars that rely on pedestrian detection using complex Deep Learning (DL) models to perform fast real-time inference has harsh requirements with regard to both provided accuracy (i.e., accurate predictions) and time delay. However, because compute resources deployed for EC are less powerful than that of the cloud computing, it is necessary to consider the trade-off with regard to computation, communication, and storage capacities of the edge servers. Much recent works have investigated strategies for adhering to these resource limitations while providing ample Quality-of-Service (QoS) [2]–[6].

Many emerging IoT applications rely on deep learning technology to infer knowledge from the data and make smart decisions. More recently, EC has been shown to reduce response time for deployed DL services [7]–[9]. However, DL models are computationally expensive to run, especially for resource-constrained IoT devices. Thus, there is a need for optimal offloading strategies for handling requests for DL-based services in EC. Furthermore, response/delay time is not the only crucial aspect of QoS for DL-based applications. Accuracy/loss of the provided DL-based services is also an important measure of the QoS. There is often a trade-off with regard to these two aspects of QoS (e.g., some models take longer to run). Thus, it is crucial to have optimal accuracy-time trade-off for DL-based services in complex EC platforms.

The challenge of maximally meeting user QoS expectations via offloading strategies in the edge computing systems is well-investigated in the literature. Several studies explored task offloading decisions in EC systems using different objectives, such as, minimizing task completion time [10]; minimizing both latency and chances of application failure [11]; and minimizing both End Users (EU) energy consumption and task's completion time [12], trade-off between accuracy and completion time using compression techniques in a network consisting of one user, one edge server and one cloud server [13]. Other works considered DL model accuracy while providing a service considering different criteria, such as energy consumption, computation, and network condition [7], [8]. However, there still exists a large gap in understanding the trade-offs between different conflicting QoS metrics, while making request scheduling/offloading decisions for DL applications, particularly in multi-edge and cloud systems. In this paper, we study optimal QoS-aware task offloading strategies for DL-based services in a three-tier user-edge-cloud computing platform. We cast the problem as an *integer linear programming* (ILP) problem where the constraints are inspired by the limited hardware resources available in the layer of edge clouds. We prove that our problem is NP-hard and propose an efficient greedy algorithm that provides close-to-optimal performance. We evaluate the proposed algorithm's performance w.r.t. the trade-off between provided accuracy and time delay from offloading decisions and compare its performance against several baselines. We perform this evaluation in both numerical simulations and in our own real-world edge computing test-bed. To the best of our knowledge, this is the first work to

explicitly consider such trade-offs for offloading decisions in a three-tier user-edge-cloud framework.

II. Problem Formulation

Here, we formally describe our considered EC architecture, as well as formally define our offloading problem. We consider a three-tier architecture. The topmost tier represents cloud servers that can be physically distributed, but virtually centralized. Here, we consider them as one single entity, called a "cloud server". The middle layer consists of a heterogeneous set of edge computing servers which are connected to the cloud server. We also consider that the edge servers can also directly communicate with each other, either through a back-haul or a device-to-device communication. Users represent the bottom layer in this architecture and we assume users can request different computationally-intensive services from their local EC servers. A local EC server of a user is the one that is the closest edge server to the user. For our work, we focus on service types that use a DL model to serve the user. However, the model is general to any service type. Different service types are used in this system, not all services can be placed on each edge server due to the edge device capacity constraints, but all service types could be placed on the central cloud server, since the cloud resources are much larger than those of the edge servers. Simply, we assume the central cloud has communication and computation constraints, but no storage constraints. Service types represent different DL-based tasks (e.g., image classification). We consider a set K of services available in the system and a set L of DL model types per service. Each service type $k \in K$ has $|L|$ different implementations of a DL model that could be used to serve the task with different levels of provided accuracy.

Model description. We consider a three-tier user-edge-cloud system and a set of requests N submitted to the system by the users at a given time. Each request $i \in N$ is submitted alongside a preferred accuracy and delay thresholds to be considered for that request's respective user satisfaction. A user with several requests can be modeled as multiple users with one request each. Note, we use "request" and "user" interchangeably throughout the paper. We consider a set of servers M that can serve user requests. Each server $j \in M$ has known computation and communication capacities, represented by γ_j and η_j, respectively. Many papers modeling EC consider storage capacity constraints; we do not consider storage constraints in this paper as the assumption is that service/model placement decision is already made. Unlike most state-of-the-art papers, we assume the cloud has limited resources — albeit its resources are more powerful than the edge servers. This is because the scale of the edge computing systems in real-world scenarios are very diverse. Cloud servers could be devices like desktop computers interacting with more resource-constrained edge devices than a large-scale remote cloud with seemingly unlimited resources. Thus, we directly and explicitly consider the resource capacities of the cloud server to allow our model to be more general and more realistic. Additionally, our approach *allows for the* consideration of more than one cloud server in the topmost layer.

For our problem, we do not explicitly distinguish between edge servers and cloud servers because both can be modelled in the same way. The main difference is on the allocated hardware resources, i.e., cloud servers have significantly more resources. Also users cannot communicate with the cloud directly, and they can only communicate with the cloud through their local edge server. Thus, there is no user request submitted to the cloud directly. It is assumed that at the start of each time-step, all servers $j \in M$ report their remaining computation and communication capacities along with the user requests they received. Therefore, the available capacity of all servers $j \in M$ and the set of user requests is assumed to be known by each server at every given time step. For simplicity, we do not consider the overhead of sending such control information in the system, as this is out of the scope of this paper. It is assumed that the user requests are submitted to their local edge servers. The goal of our problem is to make optimal request scheduling decisions, i.e. which server should serve a given request sent by a user to the user's local edge server. Note that for every request submitted to an edge server, one of the following scheduling decisions can be made: (a) the edge server serves the request locally, (b) the edge server offloads the received request to another available server (this includes both central cloud or other edge servers), or (c) the edge server drops the request entirely. At the start of each time-step, each edge server makes a scheduling decision for each service request submitted by its associated users. The edge server covering (i.e., directly associated with) user i is denoted by s_i, and it is the edge server which initially receives user i's request. To review, when a user submits a request i, they are requesting some service $k \in K$. They also submit minimum required accuracy A_i, and maximum tolerable completion time C_i to satisfy request i. A user will be satisfied only in the condition that the DL model scheduled to serve it fulfills both of its requested accuracy and completion time requirements.

Completion time. Users' requests experience different delays based on the decision to offload or process locally. Each request may experience the following types of delay throughout the system: *queuing delay*, *processing delay*, and *communication delay*. There usually exist two queues for each edge server: one for admission control (accepting requests into the system) and one for processing the services already admitted by an edge server. For this work, we assume that the delay due to the second queue is negligible, however, we explicitly consider the admission control queuing delay in our problem formulation. Since the focus of this work is on finding the optimal request scheduling decision at the edge servers, we ignore the communication delay created by sending the requests (and their dependencies) from users to the edge servers and receiving the results from the edge servers, as this delay will be the same for any decision made (e.g., offloading vs. local processing) at the edge servers. We will also assume that the delay caused by running decision algorithms at the edge servers is negligible. We will discuss the running time of

our proposed decision making algorithm in Section. III. Now we formally define each type of delay: ***1) Queuing Delay*** (T^q_{ij}) is the time between the arrival of request i to edge server j and when a decision regarding how to process the request is made on that edge server. We assume a time-slotted scenario and at each time slot, the requests 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. ***2) Processing Delay*** (T^{proc}_{ijkl}) is the time that will take for the request i to be processed on server. It depends on the service type k, DL model l used for that service, and the processing power of edge/cloud server j (CPU, GPU, RAM) processing the request. ***3) Communication Delay*** ($T^{comm}_{ijj'}$) is the communication delay needed to send request i from edge server j to another server j' in case that an offloading decision is made. $T^{comm}_{ijj'}$ is computed based on our testbed results which is discussed in section IV. Now, we define completion time of serving request i at the j-th server for service type k using DL model type l as following: if the process is offloaded, the completion time will be calculated as $c_{ijkl} = T^{comm}_{is_ij} + T^q_{is_i} + T^{proc}_{ijkl}$ and if it is done locally, the completion time will be computed as $c_{ijkl} = T^q_{is_i} + T^{proc}_{ijkl}$, where s_i represents request i's covering edge server.

MUS problem definition. Now, we formally define our proposed problem that we call the *Maximal User Satisfaction (MUS)* on the Edge problem. The goal of this problem is to maximize the expected user satisfaction based on the QoS requirements submitted alongside each service request. We cast the MUS problem as an ILP problem. First, we introduce our definition of user satisfaction.

Definition II.1 (User Satisfaction (US))**.** *We consider a user to be satisfied if, and only if, both the provided accuracy is equal or more than the user's requested accuracy and the provided completion time is equal or less than the user's requested delay. We formally define the US function for request i as:*

$$US_{ijkl} = w_{ai}\Big(\frac{a_{ijkl} - A_i}{Max_{as}}\Big) + w_{ci}\Big(\frac{C_i - c_{ijkl}}{Max_{cs}}\Big) \qquad (1)$$

where US_{ijkl} is the user satisfaction provided by serving request i at the j-th server using DL model type l of the requested service type k. C_i and A_i are the completion time and accuracy thresholds asked by user for request i. a_{ijkl} and c_{ijkl} represent the accuracy and the completion time of serving request i at server j using service type k and model type l, respectively. Max_{as} is the maximum possible provided accuracy in the system and Max_{cs} is the maximum (worst case) completion time of a task in the system. It is assumed that these values are known. Additionally, $0 \leq w_{ci} \leq 1$ and $0 \leq w_{ai} \leq 1$ are the weights that the user would assign to the requested delay and requested accuracy, respectively. These weights can be used to model different importance levels and priorities for accuracy and task's completion time. For example, some users may care more about getting more accurate results and they can tolerate some levels of delay.

Our problem considers a set of binary decision variable $\mathbf{X} \triangleq (X_{ijkl})\ \forall i \in N, j \in M, k \in K, l \in L$ where $X_{ijkl} = 1$ if and only if request i is served by device j using service k and DL model l, 0 otherwise. Below is our proposed ILP formulation:

$$\text{Max: } \frac{1}{|N|}\Big(\sum_{i\in N, j\in M}\sum_{k\in K, l\in L} US_{ijkl} X_{ijkl}\Big) \qquad (2)$$

$$\text{s.t.: } \sum_{j\in M, k\in K, l\in L} X_{ijkl} \leq 1, \qquad \forall i \in N, \qquad (2a)$$

$$X_{ijkl}\, a_{ijkl} \geq X_{ijkl}\, A_i, \qquad \forall i \in N, j \in M, k \in K, l \in L, \qquad (2b)$$

$$X_{ijkl}\, c_{ijkl} \leq X_{ijkl}\, C_i, \qquad \forall i \in N, j \in M, k \in K, l \in L, \qquad (2c)$$

$$\sum_{i,k,l} X_{ijkl}\, v_{ijkl} \leq \gamma_j, \qquad \forall j \in M, \qquad (2d)$$

$$\sum_{i,k,l}\sum_{j'\neq j} I_{ij}\, X_{ij'kl}\, u_{ijkl} \leq \eta_j, \qquad \forall j \in M, \qquad (2e)$$

$$X_{ijkl} \in \{0,1\}, \qquad \forall i \in N, j \in M, k \in K, l \in L, \qquad (2f)$$

Where, I is an indicator vector such that $|I| = |N| \times |M|$ and $I_{ij} = 1$ $(\forall i \in N, j \in M)$ if and only if (user) request i is directly covered by edge device j, and 0 otherwise (i.e. $I_{ij} = 1$ where $j = s_i$). u_{ijkl} and v_{ijkl} are communication cost and computation cost of serving request i on server j for service type k using model type l, respectively; and γ_j and η_j are the communication and computation capacity of server j. Constraint (2a) guarantees that each request should be served in just 1 server using 1 service and 1 DL model, or it will be dropped. Constraint (2b) ensures if request i is served by the j-th device, its provided accuracy is at least as large as its requested accuracy. Constraint (2c) guarantees that the completion time of serving the request i has to be less than or equal to user's requested delay for request i. Constraints (2d) and (2e) ensure that the total computation or communication costs needed to process or offload all requests coming to device j must not be more than the overall computation and communication capacity of that device, respectively. In this optimization problem, there exist three possible scheduling choices: 1) *Local processing.* The request will be served on the edge server; 2) *Offloading.* The request will be offloaded to either cloud server or one of the neighboring edge servers; 3) *Drop.* The request will not be served and will be dropped.

Special case. Although the proposed formulation considers strict QoS requirements, we can define other cases as a special case of this problem. For instance, by relaxing constraints (2b) and (2c), our problem can model scenarios where users QoS requirements are more of a suggestion rather than a hard constraint. This means that a user can be served even if its QoS requirements are not strictly met.

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

Proof. We prove Theorem 1 by a reduction from the NP-hard Maximum Cardinality Bin Packing (MCBP) problem to our problem. We are given m bins of identical capacity C and a set of $N = \{1, 2, ..., n\}$ items of weights (size) $p_i (i = 1, ..., n)$. The objective is to maximize the number of items packed

into the m bins without exceeding bin capacities and without splitting items [14]. The decision making variable is defined as $x_{ij} = 1$ if item i is placed into bin j, and zero otherwise. We will show that a simple instance of our problem would be as hard as the MCBP problem. We now construct an instance of our problem. For each item $i \in n$ construct a request r_i with total cost of getting satisfied equal to $u_i + v_i = p_i$, given that the communication and computation costs of serving request i on any server will be equal, i.e., $u_i = u_{ijkl}, v_i = v_{ijkl}, \forall j, k, l$. We also construct m servers with m bins. We consider that all edge servers and cloud servers have identical capacities equal to C equal to $\gamma_j + \eta_j = C \ \forall j$. The goal is to maximize the number of items/requests which can be placed into the m edge servers/bins of identical capacities. We claim that the optimal solution to the constructed instance of our problem gives the optimal solution to the MCBP problem. This is because an algorithm that solves our problem can solve the MCBP. Since MCBP is NP-hard, this concludes our proof [15]. □

III. Proposed Greedy Algorithm

Since MUS problem is NP-Hard (Theorem 1), we propose a greedy algorithm which we call *GUS* that attains near-optimal results w.r.t. to the objective function defined in Eq. (2). At each round, a request is served using the following rule: The GUS algorithm calculates the US of each *candidate server* which has service k and model type l for serving request i, and has enough computation capacity to serve this request, and is able to satisfy the request's minimum requirements to be a candidate server; meaning that the expected accuracy of its model should be greater than or equal to the requested accuracy and the expected total completion time provided by this server at the time of decision should be less than or equal to the requested delay. The best server with the highest US which has enough capacity to serve request i will be then selected. If request i is going to be offloaded, the communication capacity of the server that covers request i (s_i) should be also available. If there is enough capacity, the request will be assigned to server j; otherwise the algorithm will check next best candidate server with next highest US. If there is no server which can satisfy the request, it will be dropped. At the end of each round, the algorithm updates the remaining capacity for each edge cloud. The algorithm will repeat this process for all the requests. The pseudo-code for the proposed GUS algorithm is provided in Algorithm 1. The complexity of finding optimal solution for the MUS problem in the worst case is of $O((|L||M|)^{|N|})$. The running time of the GUS algorithm in the worst case is $O(|N|((|L||M|)^2 + |M| + |L||M|)) = O(|N|((|L||M|)^2))$.

Algorithm 1: Proposed Greedy Algorithm (GUS)

Input : Given N, M, K, L, I, Max_{as}, Max_{cs}, A, C
Output: Find X_{ijkl} for each request i

```
foreach request i ∈ Requests do
    s_i ← {j | I_ij = 1}
    foreach server j ∈ sorted servers having service k based on higher US do
        if c_ijkl ≤ C_i and a_ijkl ≥ A_i and v_ijkl ≤ γ_j then
            if j = s_i then
                X_ijkl ← 1
                Locally process request i
                update γ_j
                break
            else if u_ijkl ≤ η_{s_i} then
                X_ijkl ← 1
                Offload request i to place j
                update γ_j and η_j
                break
```

IV. Results

We first validated the performance of our proposed algorithm by comparing the results obtained by GUS and that of the optimal solver (computed by IBM ILOG CPLEX v 12.10.0.0) for small test cases. Our results confirm that the proposed algorithm performs close-to-optimal solution (achieving in average 90% of the optimal value) in terms of overall user satisfaction for the scenarios we have tested. The results of such comparison is omitted due to space.

Baseline algorithms. We also provided five baseline algorithms for making a comparison between our proposed greedy algorithm and other possible solutions: *1) Random-Assignment* where one of the servers (edge/cloud servers) would be selected randomly. If it can satisfy the user requirements and there is enough capacity, it will serve the request; otherwise, the request will be dropped. *2) Offload-All* where sends all the requests from edge servers to the cloud servers to serve. *3) Local Processing-All* that chooses only the local edge server for serving each of the requests. *4) Happy-Computation* in which we assume that there is no limit on the edge servers for the computation capacity and we relax the computation constraint (2d) in the optimization model. *5) Happy-Communication* where we assume no limit on the edge servers' communication capacity and we relax the communication constraint (2e) in the optimization model.

Numerical Results. In the following scenarios, we used nine edge servers and one cloud server for the test. The communication delay between the edge servers comes from different communication delays that we got from our testbed in different situations in which the bandwidth is equal to 600 bytes/msec on average. The average delay between the cloud server and edge server is based on our testbed. To account for edge servers heterogeneity, we assume that there are three types of edge servers in the system which differ based on their storage, communication, and computation capacities. Additionally, we run each test for 20000 Monte-Carlo runs and report the average. We set $|N| = 100$, $|M| = 10$, $|K| = 100$, $|L| = 10$, and it is assumed that services are randomly placed on the edge servers based on their associated storage capacity.

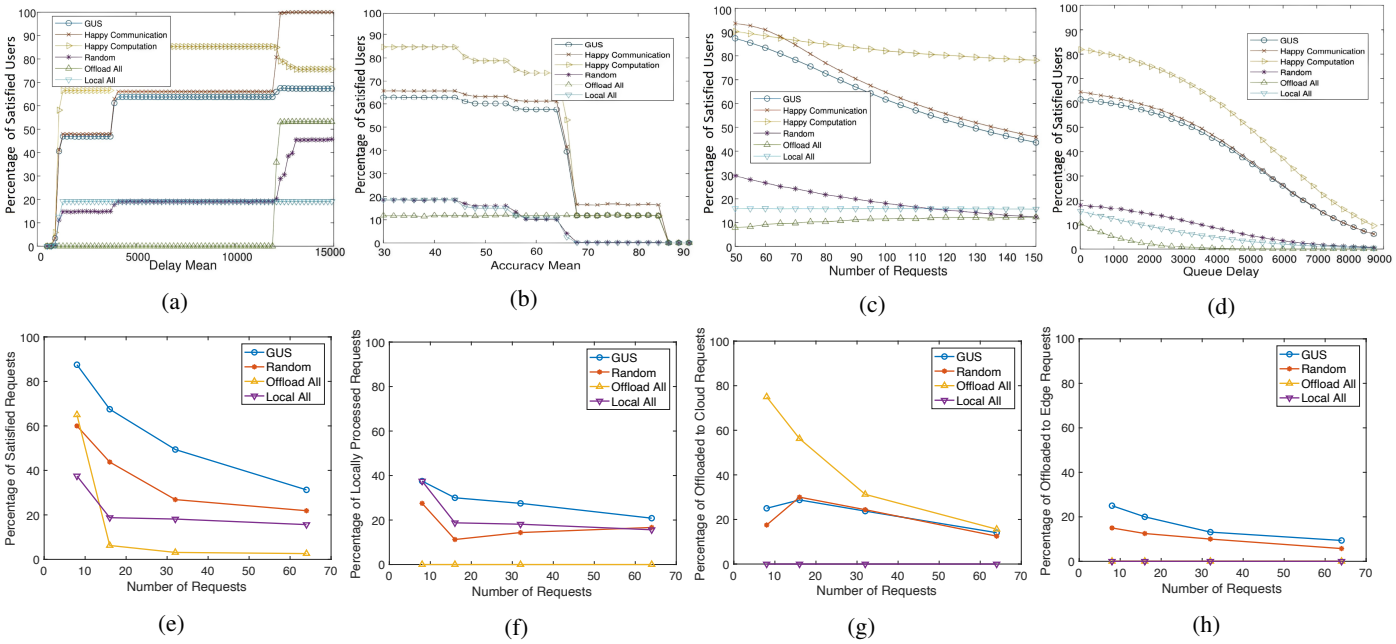

Fig. 1: Performance Evaluation Results: Numerical Results (a)-(d); Testbed Results (e)-(h)

We investigated the effect of changing requested delay and requested accuracy in the proposed system. The processing delay of running DL models is based on our testbed results which is between 950 to 1300 (msec) for edge servers, and 300 (msec) for the cloud server. The requested accuracy (A_i) and the requested delay (C_i) are both generated according to a normal distribution of $\mathcal{N}$(45%,10%) and $\mathcal{N}$(1000,4000) msec, respectively. T^q is a random number between 0 and 50 generated according to a uniform distribution. The Max_{as} and Max_{cs} in the system are 100% and 12000 (msec), respectively. We assume $w_{ai} = w_{ci} = 1$, so equal weights will be considered for both requested accuracy and requested delay. As shown Fig. 1(a), when the requested delay range increases, the total served requests will increase. The reason is that there are more requests which could be sent to the cloud server. When the requested accuracy increases, the number of satisfied requests decreases. The reason is that there would not be a DL model on the edge server which can provide the accuracy that user asked (Fig. 1(b)). When the number of requests increases, the satisfied users percent decreases (Fig. 1(c)). The reason is each EC server has limited capacity. Finally, when the requests are received at the edge servers, they will be kept in the queue until making decision and serving them. The average queue delay is a function of queue length. When the queue delay increases, the number of satisfied users decreases (Fig 1(d)). This is due to completion time exceeding requested delay. Thus, we drop them due to being unable to satisfy the requests.

Testbed Implementation. We also evaluate our solutions using a real-world testbed in the NETSCIENCE [1] laboratory, comprised of several different devices to support resource and device heterogeneity. A Linux desktop (Intel core i5-3470-3.20 GHz, RAM 8GB) serves as the central cloud server. It is connected to a NetGear R6020 router, separated by roughly 6 meters. To imitate the delay between the cloud and edge serves, we connected the router to a Raspberry Pi (RP) 3B (quad core, RAM 1GB) as a forwarder between the router and the edge servers — it is roughly 10 meters away from the NetGear router. We then have two RP 4s (quad core, RAM 4GB) which serve as our edge servers for our testbed. They are connected to the forwarder and are placed on a different floor than the forwarder in the same building, roughly 11 meters away from the forwarder. The user devices are represented by two RP 3Bs which are physically close to our edge servers, at being placed within 1 meter from the edge servers. The services we consider in our testbed are two pre-trained models: *GoogleNet* [16] (exclusively available on the cloud) and *SqueezeNet* [17] (placed on edge servers, but provides poorer accuracy and resource cost). The data we use for submitting service requests for image classification is provided by the ImageNet dataset [18]. We implement a simple management program in C++ that runs the decision algorithms and makes offloading decisions w.r.t. user requests. Common communication errors had to be addressed as well (but for brevity we do not elaborate on them here).

Testbed Results. For our testbed results, users requested thresholds on completion time and accuracy are set at $C_i = 53000$ (msec) and the $A_i = 50\%$ for all requests. We used a fixed number equal to one for both the w_{ai} and the w_{ci} $\forall i \in N$, a fixed length of 4 for the queue, and the length of each time frame to 3000 (msec), which is essentially representing how often we run a decision algorithm if the queue is not full. We used multi-threading on the edge servers and we set the

[1]https://github.com/khamfroush-lab/Fed-MEC/

processing capacity equal to three as the maximum number of threads that can run the image classification in each time-step. The communication capacity is equal to ten for each edge server. This means that each edge server can only send 10 images to other servers in a time slot. We repeated each test for two hours to remove the effects of randomness due to the wireless channel and averaged the results. When running our decision algorithms, we compute the expected processing delay based on the data we first collected from our testbed. As mentioned, the processing delay of running SqueezeNet on Raspberry 4 is 1300 (msec), on average. And, the processing delay of running GoogleNet on the cloud server is equal to 300 (msec). Given the dynamics of the real-world scenarios and the fact that the wireless channel is changing all the time, we update the value of the expected communication delay used in our GUS algorithm according to the following simple rule. The algorithm starts with an initial estimated value for the expected communication delay based on our previous observations and historical data. At each iteration, this value is updated based on the observations made from the previous round. Given the expected communication bandwidth B at time t, we compute the new expected communication bandwidth at time $t+1$ as $E[B_{t+1}] = \frac{B_t+B_{t-1}}{2}$. Using the expected bandwidth, we can then compute the expected communication delay for each image based on the expected bandwidth and the size of the image. For our tests, we started with $B = 600$ bytes/msec for the expected communication bandwidth between the edge and the cloud. We may also have to adapt the Max_{cs} parameter given that the value of expected completion time is updated. The results of the testbed implementation are shown in Fig 1. We increased the total number of requests sent to the EC system and observed the performance of GUS, and three heuristics namely, "random", "local all" and "offload all" heuristics. Fig 1(e) shows the average user satisfied percentage proving that GUS is always providing a much better rate of US compared to the other two heuristics. In the case of the "offload all" and "local all" heuristic, by increasing the number of requests, the percentage of satisfied users is decreasing because of the bottleneck created by the communication and computation capacity of the edge servers, respectively. This shows that an optimal combination of local processing and offloading (as provided by GUS) can significantly increase the US of the system. Fig. 1(f) shows the percentage of requests locally processed, Fig. 1(g) shows the percentage of the requests offloaded to the cloud, and Fig. 1(h) shows the percentage of requests that are offloaded to the other edge servers. Comparing the results provided in Fig 1, observe the percentage of the users satisfied using GUS is on average 50% more than that of the heuristics.

V. Conclusion

In this paper, we proposed a new user satisfaction (US) metric that considers the trade-off between accuracy and delay of the provided service for DL services in EC systems. We pose an offloading optimization model to maximize the proposed US given a set of capacity constraints on the edge servers and proved that this is NP-hard. Hence, we came up with a greedy algorithm, called GUS, to solve the model in polynomial-time. Finally, we evaluate on a real EC testbed consisting of devices of varying resources. The implementation results also prove the effectiveness of our proposed algorithm in comparison with baseline heuristics. Our future work will focus on considering different priorities for the requests and impacts of user mobility on the provided trade-off.

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

[2] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, 2017.

[3] T. He, H. Khamfroush, S. Wang, T. La Porta, and S. Stein, "It's hard to share: joint service placement and request scheduling in edge clouds with sharable and non-sharable resources," in *IEEE 38th ICDCS*, 2018.

[4] V. Farhadi, F. Mehmeti, T. He, T. La Porta, H. Khamfroush, S. Wang, and K. S. Chan, "Service placement and request scheduling for data-intensive applications in edge clouds," in *IEEE INFOCOM*, 2019.

[5] M. Turner and H. Khamfroush, "Meeting users' QoS in a edge-to-cloud platform via optimally placing services and scheduling tasks," in *ICNC*, 2020.

[6] 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.

[7] 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.

[8] N. Felemban, F. Mehmeti, H. Khamfroush, Z. Lu, S. Rallapalli, K. S. Chan, and T. La Porta, "Picsys: Energy-efficient fast image search on distributed mobile networks," *IEEE Trans.*, 2019.

[9] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning," *IEEE Network*, 2019.

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

[11] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, 2016.

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

[13] 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.

[14] K.-H. Loh, B. Golden, and E. Wasil, "Solving the maximum cardinality bin packing problem with a weight annealing-based algorithm," in *Operations Research and Cyber-Infrastructure*, Springer, 2009.

[15] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. on Networking*, vol. 24, no. 5, 2015.

[16] 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.

[17] 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.

[18] 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.