Data Sharing-Aware Task Allocation in Edge Computing Systems

Sanaz Rabinia
Dept. of Computer Science
Wayne State University
Detroit, USA
srabin@wayne.edu

Haydar Mehryar
Dept. of Computer Science
Wayne State University
Detroit, USA
hmehryar@wayne.edu

Marco Brocanelli
Dept. of Computer Science
Wayne State University
Detroit, USA
brok@wayne.edu

Daniel Grosu
Dept. of Computer Science
Wayne State University
Detroit, USA
dgrosu@wayne.edu

***Abstract*—Edge computing allows end-user devices to offload heavy computation to nearby edge servers for reduced latency, maximized profit, and/or minimized energy consumption. Data-dependent tasks that analyze locally-acquired sensing data are one of the most common candidates for task offloading in edge computing. As a result, the total latency and network load are affected by the total amount of data transferred from end-user devices to the selected edge servers. Most existing solutions for task allocation in edge computing do not take into consideration that some user tasks may actually operate on the same data items. Making the task allocation algorithm aware of the existing data sharing characteristics of tasks can help reduce network load at a negligible profit loss by allocating more tasks sharing data on the same server. In this paper, we formulate the data sharing-aware task allocation problem that make decisions on task allocation for maximized profit and minimized network load by taking into account the data-sharing characteristics of tasks. In addition, because the problem is NP-hard, we design the DSTA algorithm, which finds a solution to the problem in polynomial time. We analyze the performance of the proposed algorithm against a state-of-the-art baseline that only maximizes profit. Our extensive analysis shows that DSTA leads to about 8 times lower data load on the network while being within 1.03 times of the total profit on average compared to the state-of-the-art.**

***Keywords*-Edge Computing, Data sharing, Task Allocation, Profit Maximization, Network Load Minimization.**

I. Introduction

Edge computing facilitates the operations of nearby resource-limited mobile devices such as smartphones, tablets, autonomous mobile robots, drones, and connected vehicles at lower transmission latency compared to the cloud. In fact, many data-driven applications running on mobile devices need computational support to analyze locally-acquired sensor data (e.g., a video or an image from camera, an audio trace from microphone). Typical tasks include face recognition [3], image classification [14], and object tracking [19]. To offload a task, each device must transmit all the data items to be analyzed (e.g., camera frames) to one of the nearby available edge servers. On the other hand, given the possibly large number of end-user devices in the edge system and the even larger number of requests, it is important to ensure the scalability of edge resources with respect to the number of tasks and data being offloaded.

Task allocation in edge computing has been intensively studied during recent years. Due to the limited computing/energy availability of end-user devices, a significant proportion of related work has focused on offloading task execution to edge servers for lowering end-user energy consumption at a maximum latency requirement [5]–[7], [10], [17], [20]. Other studies have focused on maximizing the quality of service for end-users via task offloading within edge resource constraints [1], [2], [11], [15]. In some cases, edge servers or nearby users may receive some form of profit to provide edge resources for task offloading. Thus, some studies have focused on the topic of finding the best task allocation strategy that maximizes a defined profit in edge systems [13], [21]–[23]. Most of the above studies simply consider the transmission time of data items associated with each offloaded task on the total offloading latency estimation. Some studies provided a more accurate consideration of network packet scheduling for allocating cooperative tasks on edge servers [4], [16], [18]. However, to the best of our knowledge, none of the above solutions have considered the fact that multiple tasks from the same user may have to analyze the same data item. For example, the same camera frame can be used by a task for face recognition and by another task for object detection. Thus, allocating those tasks to different servers without considering that they share data items may lead to the necessity to send the same data item to both servers. On the other hand, allocating those tasks to the same server can help reduce the network load since only one copy of that shared data item needs to be transmitted.

In this paper, we formulate the *data sharing-aware task allocation problem* as a bi-objective mixed-integer multilinear program that maximizes the profit derived from executing tasks and minimizes the network load by taking into account the data sharing characteristics of tasks. Because this problem is NP-hard, we design a greedy algorithm, called DSTA (Data Sharing-Aware Task Allocation), that finds a feasible solution in polynomial time. Specifically, DSTA considers the task data sharing characteristics expressed as a task-data matrix to decide which tasks to allocate on the edge servers by iteratively selecting a subset of tasks that share the highest amounts of data with the already allocated tasks. In each iteration, DSTA maximizes the profit by prioritizing

high-profit/light-workload tasks for allocation to the most suitable edge server.

In summary, this paper makes the following contributions:

- To the best of our knowledge, our work is the first to exploit the key intuition that, in edge systems, multiple data-driven tasks from each user device may share some data items. This intuition can be exploited to reduce the network load in edge computing systems by allocating high data-sharing tasks to the same servers.
- We formulate the data sharing-aware problem as a bi-objective mixed-integer multilinear program that jointly maximizes the task allocation profit and minimizes the network load. We develop a novel analytical model for capturing the sharing among tasks and use it to derive the objective function that corresponds to the second objective of the problem.
- The formulated data sharing-aware problem is NP-hard. Thus, in order to provide a feasible solution in polynomial time, we design a greedy algorithm, called DSTA, that considers the tasks' data-sharing characteristics and iteratively allocates them on edge servers to maximize the profit and minimize the network load.
- We compare our proposed DSTA algorithm with a state-of-the-art baseline that only maximizes task execution profit (i.e., P-Greedy). Our results show that DSTA can reduce network data load by about 8x on average at a negligible profit loss compared to P-Greedy.

The rest of the paper is organized as follows. Section II formulates the data sharing-aware task allocation problem. Section III describes the DSTA algorithm. Section IV presents the experimental results. Section V describes the limitations of our current design and our future work. Section VI concludes the paper.

II. Data Sharing-Aware Task Allocation Problem

We consider an edge computing system composed of a set $\mathcal{S} = \{S_1, S_2, \ldots, S_M\}$ of M distributed servers, where each server S_j has a limited capacity C_j of computational resources (i.e., CPU cycles). These edge servers serve a set $\mathcal{T} = \{T_1, T_2, \ldots, T_N\}$ of N tasks originating from end-user devices. The set of tasks $\mathcal{T}$ has an associated set of data items, $\mathcal{D} = \{D_1, D_2, \ldots, D_K\}$, that are needed to execute the tasks. We denote the size of data item D_k by d_k, $k = 1, 2, \ldots, K$. Each task T_i is characterized by a tuple $(r_i, p_i, [A]_{i,*})$, where r_i is the amount of computational resources required by T_i, p_i is the profit for executing T_i, and $[A]_{i,*}$ is the i-th row of the task-data matrix, A. The *task-data matrix* A is a $N \times D$ matrix, where $a_{ik} = d_k$, if task T_i requires data item D_k, and 0, otherwise. The tasks need to be allocated to the servers such that the total profit obtained from executing the tasks is maximized and the total amount of data transferred in the network is minimized.

Table I: Notation

Expression	Description
$\mathcal{T}$	Set of tasks.
N	Number of tasks.
T_i	Task i.
r_i	Requested amount of CPU resource by T_i.
p_i	Profit of task T_i.
$\mathcal{S}$	Set of servers.
M	Number of servers.
S_j	Server j.
C_j	CPU capacity of server S_j.
$\mathcal{D}$	Set of data items.
D	Number of data items.
D_k	Data item k.
d_k	Size of data item D_k.
A	Task-data matrix (a_{ik}; $i = 1, \ldots, N$; $k = 1, \ldots, D$).
$\sigma_{\mathcal{I}}$	Sharing parameter.
$\mathcal{T}^c$	Set of candidate tasks.
$\mathcal{D}^c$	Set of candidate data which is assigned to servers.
s_k	Sum of column k entries of matrix A.
E_i	Efficiency function for task T_i.
supp($[A]_{*,k}$)	Support of column k in matrix A.

We formulate the *data sharing-aware task allocation problem* (DSTAP) as a bi-objective mixed-integer multilinear program:

$$\text{maximize:} \quad \sum_{j=1}^{M} \sum_{i=1}^{N} p_i x_{ij} \tag{1}$$

$$\text{minimize:} \quad \sum_{\mathcal{I} \in \mathcal{P}(\mathcal{T})} (-1)^{(|\mathcal{I}|+1)} \sigma_{\mathcal{I}} \sum_{j=1}^{M} \prod_{i \in \mathcal{I}} x_{ij} \tag{2}$$

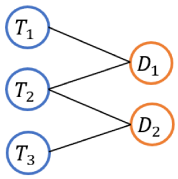
subject to:

$$\sum_{i=1}^{N} r_i x_{ij} \leq C_j, \quad \forall j \in \{1, \ldots, M\} \tag{3}$$

$$\sum_{j=1}^{M} x_{ij} \leq 1, \quad \forall i \in \{1, \ldots, N\} \tag{4}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i, \forall j \tag{5}$$

The first objective (Equation (1)) is to maximize the total profit. The decision variable x_{ij} is 1, if task T_i is allocated to server S_j, and 0, otherwise. The second objective (Equation (2)) is to minimize the total amount of data offloaded from user devices to the servers, which depends on the decision variables x_{ij} and on the data sharing among tasks. Here, $\mathcal{P}(\mathcal{T})$, is the power set of the set of indices of the tasks in $\mathcal{T}$, and $\mathcal{I}$ is an element of the power set. We define the *sharing parameter*, $\sigma_{\mathcal{I}}$, as the total amount of data shared among the tasks whose indices are in set $\mathcal{I}$. In the next paragraph, we give more details on how the sharing parameter is computed and explain how the cost function (2) captures the sharing of data and gives the total amount of data in the network. Constraint (3) ensures that the total allocated computational requests to a server does not exceed the capacity of the server. Constraint (4) ensures that each



$$\begin{array}{c} \\ T_1 \\ T_2 \\ T_3 \end{array} \begin{array}{c} \begin{array}{cc} D_1 & D_2 \end{array} \\ \begin{bmatrix} 4 & 0 \\ 4 & 6 \\ 0 & 6 \end{bmatrix} \end{array}$$

Figure 1: DSTAP instance with three tasks, two data items, and task-data matrix.

task is allocated to only one server, while Constraint (5) guarantees the integrality of the decision variables. Table I, summarizes the notation used in the paper.

To explain how the sharing is captured in equation (2), we use a small example consisting of a set of three tasks $\mathcal{T} = \{T_1, T_2, T_3\}$, a set of two data items $\mathcal{D} = \{D_1, D_2\}$, and a set of two servers $\mathcal{S} = \{S_1, S_2\}$. For this example, we consider that T_1 needs D_1, T_2 needs D_1 and D_2, and T_3 needs D_2. The task-data matrix A and the associated bipartite graph capturing the sharing for this example is given in Figure 1. One partition of the bipartite graph consists of vertices corresponding to the tasks, while the other partition consists of vertices corresponding to the data items. We now show how the sharing parameter $\sigma_{\mathcal{I}}$ used in Equation (2) is determined. The sharing parameter $\sigma_{\mathcal{I}}$ is the amount of data shared by the tasks whose indices are in set $\mathcal{I}$. For our example, we have $\sigma_1 = 4$, $\sigma_2 = 10$, $\sigma_3 = 6$, $\sigma_{12} = 4$, $\sigma_{13} = 0$, $\sigma_{23} = 6$, and $\sigma_{123} = 0$. Suppose that T_1 and T_2 are allocated to S_1, and T_3 is allocated to S_2, then we have $x_{11} = x_{21} = x_{32} = 1$ and $x_{12} = x_{22} = x_{31} = 0$.

The total amount of data offloaded from users to the servers (according to the second objective function) is given by:

$$\begin{aligned} &(+1)\,[\sigma_1(x_{11} + x_{12}) + \sigma_2(x_{21} + x_{22}) + \sigma_3(x_{31} + x_{32})] + \\ &(-1)\,[\sigma_{12}(x_{11}x_{21} + x_{12}x_{22}) + \sigma_{13}(x_{11}x_{31} + x_{12}x_{32}) \\ &\quad +\sigma_{23}(x_{21}x_{31} + x_{22}x_{32})] + \\ &(+1)\,[\sigma_{123}(x_{11}x_{21}x_{31} + x_{12}x_{22}x_{32})] \end{aligned}$$

Plugging in the values of x_{ij} and $\sigma_{\mathcal{I}}$ in the above equation we obtain: $\sigma_1 + \sigma_2 + \sigma_3 - \sigma_{123} = 4 + 10 + 6 - 4 = 16$. We can easily check that the total amount of data offloaded to servers is 16. Since T_1 and T_2 are assigned to the same server both D_1 and D_2 need to be offloaded to server S_1 and the amount of data offloaded to S_1 is 10. T_3 is assigned to server S_2 and it needs only D_2, thus the total amount of data offloaded to S_2 is 6. Therefore, the total amount of data offloaded in the system is 16.

The Knapsack problem is a special case of DSTAP, that is, by removing the second objective from DSTAP, the problem becomes the Knapsack problem. The Knapsack is a known NP-hard problem [9] and since it is a special case of DSTAP, we have that DSTAP is NP-hard. Thus, there is no polynomial algorithm that obtains an optimal solution of DSTAP, unless P = NP. Therefore, in the next section, we design a greedy algorithm that finds a feasible solution in polynomial time.

III. DSTA Algorithm

We design a greedy algorithm, called DSTA, for solving the problem. DSTA takes into account the sharing characteristics of the tasks when deciding which tasks to allocate on the edge servers. That is, it iteratively selects a subset of tasks that share the highest amounts of data with the tasks that are already allocated. In each iteration, it establishes a greedy order among these tasks, that is induced by a function that prioritizes high-profit and light-workload tasks for allocation to the most suitable edge server. DSTA is given in Algorithm 1. The input of DSTA consists of the set of tasks, $\mathcal{T}$; the set of servers, $\mathcal{S}$; and the task-data matrix, A.

Initialization (Lines 1-8). DSTA initializes the set of candidate tasks $\mathcal{T}^c$ and the set of candidate data items $\mathcal{D}^c$ to the empty set, and the allocation matrix X to zero (Lines 1 to 3). Here, the allocation matrix has as entries the variables x_{ij}, where $x_{ij} = 1$, if task T_i is allocated to server S_j, and 0, otherwise. Next (Line 4), it sorts the servers in non-increasing order of their capacities. The ordering of the servers after sorting is given by the permutation $\beta(j)$.

DSTA uses an array s, whose entries $s_k = \sum_{i=1}^{N} a_{ik}$, for $k = 1, \ldots, K$. That is, s_k is the sum of the entries of column k of task-data matrix A. Since the column k of A corresponds to data item k, s_k is the total amount of data item k needed by the tasks without considering sharing. DSTA computes the entries of s in Lines 6 to 8.

Allocation Strategy Overview. The while loop in Lines 9 to 39 is executed until the set of tasks $\mathcal{T}$ becomes empty. In each iteration of the loop, the algorithm determines which data item is the most shared and the tasks that share it, computes the efficiency metric that is used to establish the greedy order among those tasks, and allocates the tasks to servers in the order given by the greedy order. In the following, we described these operations in more details.

Data Sharing Analysis (Lines 10-20). The algorithm uses an additional array s' whose entry s'_k is set to 1 if a task in the candidate set is using the data item D_k and D_k has not been assigned to a server yet (Lines 11 to 15). The algorithm determines the *support* of the array s', denoted here by $\text{supp}(s')$, which is defined as the set of all indices corresponding to nonzero entries in s' (Line 16). If the size of the support of s' is greater than zero, there are data items that have not been assigned to servers yet, thus the algorithm places in set $\mathcal{K}$ the indices of the data items that have not been assigned yet and have the largest value of s_k (Line 17). If the size of the support of s' is equal to zero, then no data item was allocated yet (i.e., this is the first iteration of the while loop). Thus, DSTA places in set $\mathcal{K}$ the indices

Algorithm 1 DSTA Algorithm

Input: $\mathcal{T}$: set of tasks;
$\mathcal{S}$: set of edge servers;
A: task-data matrix.
1: $\mathcal{T}^c \leftarrow \emptyset$
2: $\mathcal{D}^c \leftarrow \emptyset$
3: $X \leftarrow [0]$
4: Sort servers in non-increasing order of capacity C_j
Let $S_{\beta(1)}, S_{\beta(2)}, \ldots, S_{\beta(M)}$ be the order
5: $s \leftarrow [0]$
6: **for** $k = 1, \ldots, D$ **do**
7: **for** $i = 1, \ldots, N$ **do**
8: $s_k \leftarrow s_k + a_{ik}$
9: **while** $|\mathcal{T}| > 0$ **do**
10: $s' \leftarrow [0]$
11: **for** $i = 1, \ldots, N$ **do**
12: **if** $T_i \in \mathcal{T}^c$ **then**
13: **for** $k = 1, \ldots, D$ **do**
14: **if** $D_k \notin \mathcal{D}^c$ **and** $a_{ik} \neq 0$ **then**
15: $s'_k \leftarrow 1$
16: **if** $|\text{supp}(s')| > 0$ **then**
17: $\mathcal{K} = \{k | \ \forall\ l \in \{1, \ldots, D\} : \ s_l s'_l \leq s_k s'_k\}$
18: **else**
19: $\mathcal{K} = \{k | \ \forall\ l \in \{1, \ldots, D\} : \ s_l \leq s_k\}$
20: $\tilde{k} \leftarrow \text{argmax}_{k \in \mathcal{K}} \{|\text{supp}([A]_{*,k})|\}$
21: **for** $i = 1, \ldots, N$ **do**
22: $E_i \leftarrow 0$
23: **if** $a_{i\tilde{k}} \neq 0$ and $T_i \in \mathcal{T}$ **then**
24: $E_i \leftarrow \frac{p_i}{\sqrt{\frac{r_i}{\sum_{j=1}^{M} C_{\beta(j)}}}}$
25: Sort tasks in non-increasing order of E_i
Let $T_{\alpha(1)}, T_{\alpha(2)}, \ldots, T_{\alpha(N)}$ be the order
26: **for** $i = 1, \ldots, N$ **do**
27: **if** $E_{\alpha(i)} > 0$ **then**
28: **for** $j = 1, \ldots, M$ **do**
29: **if** $C_{\beta(j)} - r_{\alpha(i)} \geq 0$ **then**
30: $C_{\beta(j)} \leftarrow C_{\beta(j)} - r_{\alpha(i)}$
31: $x_{\alpha(i)\beta(j)} \leftarrow 1$
32: $\mathcal{T}^c \leftarrow \mathcal{T}^c \cup \{T_{\alpha(i)}\}$
33: $\mathcal{T} \leftarrow \mathcal{T} \backslash \{T_{\alpha(i)}\}$
34: **break**
35: **else**
36: **if** $j = M$ **then**
37: $\mathcal{T} \leftarrow \mathcal{T} \backslash \{T_{\alpha(i)}\}$
38: $\mathcal{D}^c \leftarrow \mathcal{D}^c \cup \{D_{\tilde{k}}\}$
39: $s_{\tilde{k}} \leftarrow 0$
40: **output:** X

of the items that have the largest value of s_k (Line 19). Next, DSTA determines the index $\tilde{k}$ of the items in set $\mathcal{K}$ whose corresponding column in the task-data matrix A has the largest support, i.e., the item shared by the largest number of tasks (Line 20).

Efficiency Analysis and Task Selection (Lines 21-24). In Lines 21 to 24, DSTA computes the *efficiency function* which is used to establish the greedy order among the tasks. The tasks will be considered for allocation in the order provided by this greedy order. The efficiency function is computed only for the tasks that share the data item $\tilde{k}$, which is shared by the highest number of tasks and that has the greatest corresponding total amount of data. Thus, the efficiency function is computed only for the tasks that are not allocated yet and have $a_{i\tilde{k}} \neq 0$ (Line 23). The efficiency function is defined by

$$E_i = \frac{p_i}{\sqrt{\frac{r_i}{\sum_{j=1}^{M} C_j}}}. \tag{6}$$

The efficiency function for a given task can be viewed as a density measure, computed as the profit obtained from executing the task divided by the square root of the relative size of the request. Here, the relative size of the request is with respect to the total capacity of the servers in the system. This efficiency function allows the algorithm to allocate the tasks in the order of their highest profit density and therefore obtain high values for the total profit gained from executing the tasks.

Allocation of the Selected Tasks (Lines 25-39). Once the efficiency function is determined, DSTA sorts the tasks in non-increasing order of E_i (Line 25). The ordering of the tasks after sorting is given by the permutation $\alpha(i)$. The algorithm goes over the tasks with $E_i \geq 0$ (i.e., the tasks with high data sharing for which the efficiency metric was determined) in the order given by permutation $\alpha(i)$ and attempts to allocate them to the available servers. In Lines 28 to 37, the servers are considered for task allocation in the non-increasing order of their capacities (given by the permutation $\beta(j)$). DSTA checks if the given server has enough capacity to handle the request. If it has enough capacity, the capacity of the server is decreased by the size of the request, the entry corresponding to task $T_{\alpha(i)}$ and server $S_{\beta(j)}$ in the allocation matrix X is set to 1, the task is added to the set of candidate tasks $\mathcal{T}^c$ and removed from the set of tasks $\mathcal{T}$. Once a task is allocated the algorithm exits from the for loop in Line 34. If the server does not have enough capacity the algorithm considers for allocation the next server in the order. If none of the servers have enough capacity to allocate the task then the task is removed from the set of tasks $\mathcal{T}$ (Lines 36 to 37). Finally, at the end of each while loop iteration (Lines 38 and 39), DSTA adds data item $D_{\tilde{k}}$ to the candidate data set and sets $s_{\tilde{k}}$ to 0, to avoid reconsidering data item $D_{\tilde{k}}$ in the next iterations.

Complexity of DSTA. The while loop (Lines 9 to 39) determines the time complexity of DSTA which is $\mathcal{O}(N^2(D+M))$. This is mainly due to the running time of the for loop in Lines 26 to 37, which takes $\mathcal{O}(NM)$, and the computation of $\tilde{k}$ in Line 20 which takes $\mathcal{O}(ND)$. In the worst case the while loop is executed $\mathcal{O}(N)$ times, and thus the running time of the while loop is $\mathcal{O}(N^2(D+M))$. Therefore, DSTA has a time complexity of $\mathcal{O}(N^2(D+M))$.

Algorithm 2 P-Greedy Algorithm

Input: $\mathcal{T}$: set of tasks;
$\mathcal{S}$: set of edge servers.
1: $X \leftarrow [0]$
2: Sort servers in non-increasing order of capacity C_j. Let $S_{\beta(1)}$, $S_{\beta(2)}$, ..., $S_{\beta(M)}$ be the order.
3: **while** $|\mathcal{T}| > 0$ **do**
4: $\mathcal{I} = \{i | \ \forall \ l \in \{1, \ldots, N\} : \ p_l \leq p_i\}$
5: $\tilde{i} \leftarrow \text{argmin}_{i \in \mathcal{I}}\{r_i\}$
6: **for** $j = 1, \ldots, M$ **do**
7: **if** $C_{\beta(j)} - r_{\tilde{i}} \geq 0$ **then**
8: $\mathcal{T} \leftarrow \mathcal{T} \backslash \{T_{\tilde{i}}\}$
9: $C_{\beta(j)} \leftarrow C_{\beta(j)} - r_{\tilde{i}}$
10: $x_{\tilde{i}\beta(j)} \leftarrow 1$
11: **break**
12: **else**
13: **if** $j = M$ **then**
14: $\mathcal{T} \leftarrow \mathcal{T} \backslash \{T_{\tilde{i}}\}$
15: **output:** X

IV. Experimental Analysis

In this section, we investigate the performance of the proposed algorithm, DSTA, and compare it with a baseline algorithm. We implement the algorithms in Java and run the simulation experiments on a system with 2 cores Intel i7-7660U at 2.5GHz, 16GB of memory, and 512GB SSD of storage. First, we describe the baseline algorithm used for comparison and how we generate the taskset and data sharing characteristics, and then analyze the results.

A. Baseline Algorithm

The proposed DSTA algorithm jointly maximizes the total profit and minimizes the network data load by considering the data sharing characteristics of the tasks. Therefore, for comparison purposes, we define the P-Greedy baseline algorithm that, similarly to existing work [13], [21]–[23], allocates tasks with the objective of maximizing the profit without considering the network data load.

Algorithm 2, shows the pseudo-code of P-Greedy. The algorithm has as input the task set an the set of edge servers. First, it initializes the allocation matrix X (Line 1), which is also the output of the algorithm (Line 15). Then, it sorts the servers in non-increasing order of capacity (Line 2) and allocates one by one the tasks to the available servers (Lines 3 to 14). Specifically, in each iteration of allocation P-Greedy extracts the subset of tasks with the highest profit (Line 4) and, among them, selects the one with the lowest computational requirement (Line 5). Then, it finds the server with the highest available capacity to allocate the selected task and, accordingly, updates the server capacity and the allocation matrix entries (Lines 6 to 11). If there is no server with enough capacity, the selected task is left unallocated (Lines 12 to 14).

Table II: Distribution of parameters

Parameter	Distribution/Value
Server	capacity range: $[7, 14]$ number of servers: 40
Image file size	pixel count: random$[100, 400]$ pixels width and height bit depth: random$[1, 64]$ bit
Video file size	frame size: random$[5, 1000]$ byte frame rate : random$[12, 60]$ fps time: random$[1, 20]$ seconds
Audio file size	bit depth: random$[1, 64]$ bit sample rate: random$[1, 16000]$ Hz audio length: random$[10, 1000]$ seconds channels: random$[1, 12]$ mono, stereo, quad

B. Taskset and Data-sharing Characteristics

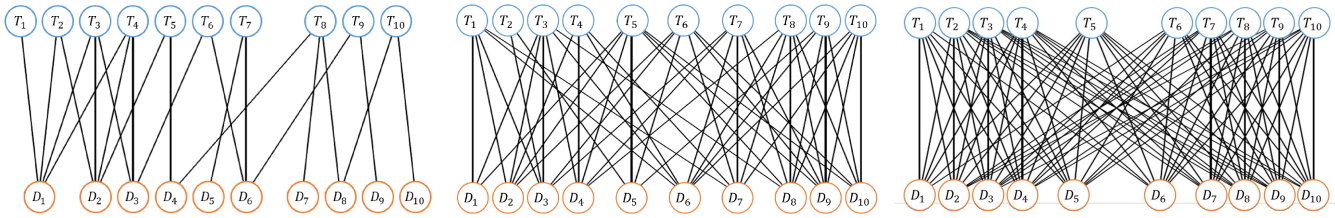
Server Capacity and Task Demands. We determine the capacity of the servers based on the total number of instructions that can be executed in a certain amount of time. We define the *demand ratio*, ρ, to characterize the relationship between the total capacity of the servers and the total amount of requests from users:

$$\rho = \frac{\sum_{j=1}^{M} C_j}{\sum_{i=1}^{N} r_i}, \quad 0 \leq \rho \leq 1 \tag{7}$$

Based on the demand ratio, we consider three different cases: *low demand* ($\rho \ll 1$), when the total requested capacity is much greater than the total available capacity of the edge servers; *average demand* ($\rho \approx 1/2$), when the total requested capacity is about half the total available capacity of the edge servers; and *high demand* ($\rho \approx 1$) when the total requested capacity is approximately equal to the total capacity available on the edge servers. We generate request sizes according to the random exponential distribution for three demand ratios 0.3, 0.1, and 0.05, corresponding to low, average, and high demand cases. The profits for 100 different tasks and 10 different problem instances are drawn from the random exponential distribution. Note, we have also tested other distributions such as uniform and noticed similar results to those described in the next section. However, for space reasons we omit those results.

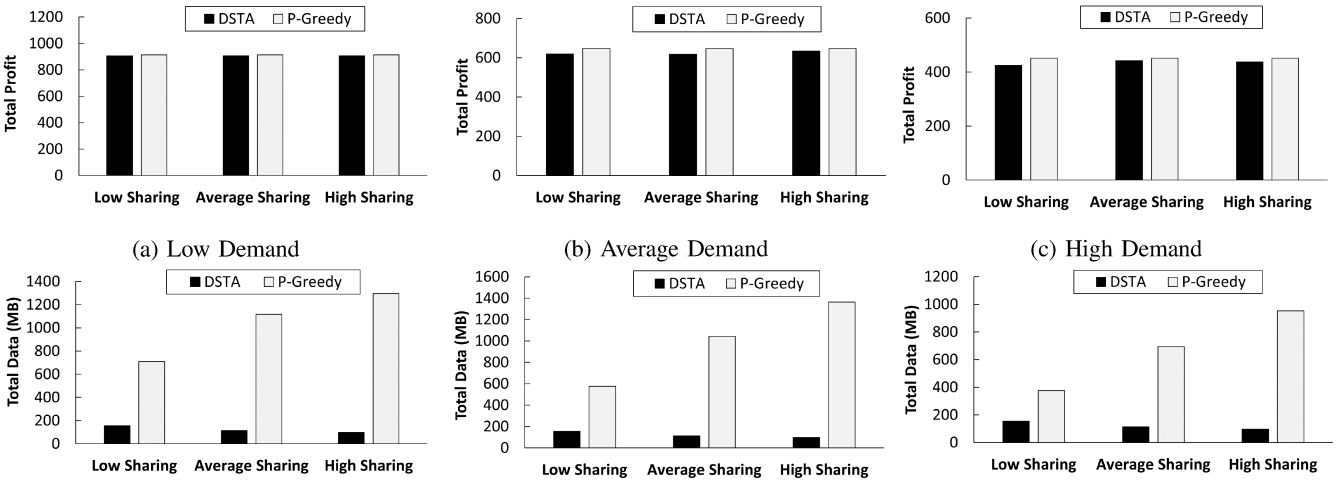
Data Items and Data Sharing. For the data items and their sizes, we consider three typical data types that can be offloaded to the edge servers for analysis: image files, video files, and audio files. We use the set of parameters listed in Table II for each data type to calculate their sizes [12]. The table also shows the ranges for each of those parameters. Using those random data sizes, we generate tasksets of 100 tasks and 100 data items to allocate on 40 servers with capacities within the interval $[7, 14]$.

We leverage the Erdös-Rényi random graph model [8] to generate the bipartite graph characterizing the data sharing pattern for the generated taskset. In the Erdös-Rényi model,



(a) Low Sharing (b) Average Sharing (c) High Sharing

Figure 2: Examples of bipartite graphs (10 tasks and 10 data items) generated using the Erdös-Rényi random graph model: (a) low data sharing, (b) average data sharing, and (c) high data sharing.



(a) Low Demand (b) Average Demand (c) High Demand

(d) Low Demand (e) Average Demand (f) High Demand

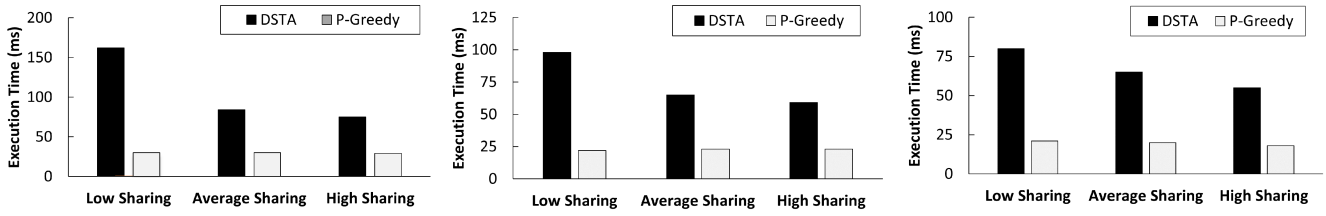
Figure 3: Total profit (a-c) and total amount of data on the network (d-f) for various combinations of workload demand and data sharing characteristics (exponential distribution) across offloaded tasks. Our proposed DSTA algorithm is able to achieve a good tradeoff between total profit and total amount of data on the network due to task offloading compared to the P-Greedy baseline.

for a graph with n vertices and m edges, the probability of generating each edge is given by: $p^m(1-p)^{\binom{n}{2}-m}$. The parameter p is a real number between 0 and 1, where larger values corresponds to a higher number of edges in the graph, i.e., a larger number of tasks share data items with each other. We implement the Erdös-Rényi model using the *igraph* R package. We generate instances for three different cases: *low sharing* ($0 \leq p \leq 0.3$) where only a few tasks share data items, *average sharing* ($0.3 < p \leq 0.6$), and *high sharing* ($0.6 < p \leq 1$) where a large number of tasks share data items. Figure 2 shows an example of a randomly generated bipartite graph for three different cases of 10 tasks and 10 data items (low sharing with $p = 0.2$; average sharing with $p = 0.5$; and high sharing with $p = 0.9$).

C. Experimental Results

To analyze the performance of DSTA, we considered nine different cases according to the demand and sharing ratios: (low, average, high) demand $\times$ (low, average, high) sharing. We consider instances of the problem with 100 tasks, 40 servers, and 100 data items with various sizes. Figure 3, shows the results obtained by DSTA and P-Greedy for the two main performance metrics, total profit and total data size in the network. Figure 4, shows the execution time of these two algorithms for the nine different cases considered here.

Profit Analysis. Figure 3a, shows the total profit obtained by DSTA and P-Greedy for the low demand case. The algorithms obtain almost the same total profit. The profit obtained by DSTA is slightly lower than that obtained by P-Greedy, that is, 0.98%, 0.78%, 0.66% of the profit of P-Greedy for low, average, and high sharing cases, respectively. Figure 3b, shows the total profit obtained by DSTA and P-Greedy for the average demand case. P-Greedy obtains a higher total profit. P-Greedy obtains a total profit that is 4.29%, 4.54%, and 1.88% higher than that



(a) Low Demand (b) Average Demand (c) High Demand

Figure 4: The execution time of the algorithms for: (a) low, (b) average, and (c) high demand and different sharing scenarios.

obtained by DSTA for low sharing, average sharing, and high sharing cases, respectively. Figure 3c, shows the total profit obtained by DSTA and P-Greedy for the high demand case. P-Greedy obtains a total profit that is 5.78%, 2.04%, and 2.9% higher than that obtained by DSTA for low sharing, average sharing, and high sharing cases, respectively. As one can notice, the total average profit generally decreases from low demand to high demand case studies. This is due to the fact that higher demand cases prevent a higher number of tasks to be allocated on the available servers. In summary, P-Greedy achieves an average profit that is only 1.03 times than that of DSTA, which proves that our algorithm can still achieve relatively high profit while also considering network data load.

Network Data Load Analysis. Figures 3d-3f, show the total data size in the network corresponding to the allocation obtained by DSTA and P-Greedy. In all three demand cases, the total size of data corresponding to the allocation obtained by DSTA is much smaller than that corresponding to P-Greedy. In Figure 3d, for low demand case, the data size corresponding to P-Greedy is 4.59, 9.82, 13.32 times greater than that of DSTA for low sharing, average sharing, and high sharing scenarios, respectively. In Figure 3e, for average demand case, the data size corresponding to P-Greedy is 3.73, 9.16, and 14 times greater than that of DSTA for low sharing, average sharing, and high sharing cases, respectively. In Figure 3f, for high demand case, the data size corresponding to P-Greedy is 2.43, 6.09, and 9.79 times greater than that of DSTA for low sharing, average sharing, and high sharing cases, respectively. In all the demand cases, by increasing the sharing ratio we observe an increase in the total data size for P-Greedy. This highlights the fact that allocating tasks sharing relatively large amounts of data without considering their sharing characteristics can rapidly lead to unnecessarily high network data load. On the contrary, the proposed DSTA algorithm actually exploits the higher data sharing characteristics during task allocation, which leads DSTA to reduce the total network load for higher data sharing. On average, DSTA allows to reduce by 8 times the total network load compared to the state-of-the-art P-Greedy baseline with, as described in previous paragraph, a minimal loss in average profit.

Execution Time Analysis. Figure 4 shows the execution time of the two algorithms. Comparing different sharing scenarios within the same demand case, we can observe that the amount of data sharing does not affect the execution time of P-Greedy. This is because the decisions of P-Greedy are not dependent on this factor. In all cases, as expected, DSTA has a higher execution time compared to P-Greedy because it does not take sharing into account, thus saving in total instructions executed. On the other hand, DSTA's execution time is still relatively low considering the relatively large test cases under consideration (i.e., < 162ms in the worst case).

As we can observe, DSTA has the largest execution time for low demand - low sharing cases and lower execution time for average and high sharing cases. The main reason for this is that DSTA spends more time to re-calculate the efficiency function after each task allocation round. For example, for high sharing cases, DSTA can assign more tasks at each allocation iteration, thus considerably reducing the number of total allocation rounds.

V. Limitations and Future Work

In this paper, we study for the first time the problem of data sharing-aware task allocation in edge computing. Given that no previous work has studied similar problems, we have preferred to limit its complexity by considering only server computational capacity as a constraint without including other existing constraints such as limited network bandwidth and limited edge storage. The consideration of these limitations would make the problem considerably more complex than the one studied in this paper. In addition, the proposed DSTA algorithm does not provide any approximation guarantee of performance.

In our future work, we plan to extend the proposed greedy algorithm to provide approximation guarantees. In addition, we plan to generalize the proposed algorithm to consider as additional constraints network bandwidth and storage, which make the data sharing-aware allocation problem a bicriteria version of the multi-dimensional knapsack problem.

VI. Conclusion

In this paper, we studied the data sharing-aware problem, which we defined as a bi-objective mixed-integer multilinear program that maximizes the profit derived from executing tasks on edge servers and minimizes the network traffic load by taking into account the data sharing characteristics of tasks. We designed DSTA, a greedy algorithm that considers the task data sharing characteristics to decide which tasks to allocate on the edge servers by iteratively selecting a subset of tasks that share the highest amounts of data with the already allocated tasks. In each iteration, DSTA maximizes the profit by prioritizing high-profit/light-workload tasks for allocation to the most suitable edge server. We analyzed the performance of the proposed algorithm against a state-of-the-art baseline that only maximizes profit. Our extensive analysis showed that DSTA leads to about 8 times lower data load on the network while being within 1.03 times of the total profit on average compared to the state-of-the-art.

Acknowledgment

This research was supported in part by the US National Science Foundation under Grant CCF-2118202.

References

[1] C. Avasalcai, C. Tsigkanos, and S. Dustdar. Decentralized resource auctioning for latency-sensitive edge computing. In *Proc. IEEE Int. Conf. on Edge Computing (EDGE)*, pages 72–76, 2019.

[2] H. Badri, T. Bahreini, D. Grosu, and K. Yang. Risk-aware application placement in mobile edge computing systems: A learning-based optimization approach. In *Proc. IEEE Int. Conf. on Edge Computing (EDGE)*, pages 83–90, 2020.

[3] A. Chen, H. Xing, and F. Wang. A facial expression recognition method using deep convolutional neural networks based on edge computing. *IEEE Access*, 8:49741–49751, 2020.

[4] Q. Chen, Z. Zheng, C. Hu, D. Wang, and F. Liu. Data-driven task allocation for multi-task transfer learning on the edge. In *Proc. 39th IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*, pages 1040–1050, 2019.

[5] X. Chen and G. Liu. Joint optimization of task offloading and resource allocation via deep reinforcement learning for augmented reality in mobile edge network. In *Proc. IEEE Int. Conf. on Edge Computing (EDGE)*, pages 76–82, 2020.

[6] Y. Chen, N. Zhang, Y. Zhang, X. Chen, W. Wu, and X. S. Shen. Toffee: Task offloading and frequency scaling for energy efficiency of mobile devices in mobile edge computing. *IEEE Transactions on Cloud Computing*, pages 1–1, 2019.

[7] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. S. Quek. Offloading in mobile edge computing: Task allocation and computational frequency scaling. *IEEE Transactions on Communications*, 65(8):3571–3584, 2017.

[8] P. Erdös and A. Rényi. On random graphs i. *Publicationes Mathematicae Debrecen*, 6:290, 1959.

[9] M. R. Garey and D. S. Johnson. *Computers and intractability, A guide to the theory of NP-Completeness*, volume 29. WH Freeman, New York, 1979.

[10] X. Huang, L. He, and W. Zhang. Vehicle speed aware computing task offloading and resource allocation based on multi-agent reinforcement learning in a vehicular edge computing network. In *Proc. IEEE Int. Conf. on Edge Computing (EDGE)*, pages 1–8, 2020.

[11] S. Jošilo and G. Dán. Decentralized algorithm for randomized task allocation in fog computing systems. *IEEE/ACM Transactions on Networking*, 27(1):85–97, 2019.

[12] S. K. Alambra, D. Miszewska. Omni calculator. https://www.omnicalculator.com/other, October 2020.

[13] A. Kiani and N. Ansari. Toward hierarchical mobile edge computing: An auction-based profit maximization approach. *IEEE Internet of Things Journal*, 4(6):2082–2091, 2017.

[14] E. Kristiani, C.-T. Yang, and C.-Y. Huang. isec: An optimized deep learning model for image classification on edge computing. *IEEE Access*, 8:27267–27276, 2020.

[15] J. Lee, H. Ko, J. Kim, and S. Pack. Data: Dependency-aware task allocation scheme in distributed edge clouds. *IEEE Trans. on Industrial Informatics*, 16(12):7782–7790, 2020.

[16] Y. Sahni, J. Cao, and L. Yang. Data-aware task allocation for achieving low latency in collaborative edge computing. *IEEE Internet of Things Journal*, 6(2):3512–3524, 2019.

[17] T. X. Tran and D. Pompili. Joint task offloading and resource allocation for multi-server mobile-edge computing networks. *IEEE Trans. on Vehicular Technology*, 68(1):856–868, 2019.

[18] H. Xing, L. Liu, J. Xu, and A. Nallanathan. Joint task assignment and wireless resource allocation for cooperative mobile-edge computing. In *Proc. IEEE Int. Conf. on Communications (ICC)*, pages 1–6, 2018.

[19] B. Yang, X. Cao, C. Yuen, and L. Qian. Offloading optimization in edge computing for deep-learning-enabled target tracking by internet of uavs. *IEEE Internet of Things Journal*, 8(12):9878–9893, 2021.

[20] S. Yang, F. Li, M. Shen, X. Chen, X. Fu, and Y. Wang. Cloudlet placement and task allocation in mobile edge computing. *IEEE Internet of Things J.*, 6(3):5853–5863, 2019.

[21] H. Yuan and M. Zhou. Profit-maximized collaborative computation offloading and resource allocation in distributed cloud and edge computing systems. *IEEE Trans. on Automation Science and Engineering*, 18(3):1277–1287, 2021.

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

[23] D. Y. Zhang and D. Wang. An integrated top-down and bottom-up task allocation approach in social sensing based edge computing systems. In *Proc. IEEE Conf. on Computer Communications (INFOCOM)*, pages 766–774, 2019.