Data Sharing-Aware Online Algorithms for Task Allocation in Edge Computing Systems

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

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

***Abstract*—Many of the tasks offloaded to edge devices perform computation to analyze sensing data. Transferring this data from end-user devices to edge servers leads to increased latency and congestion in the edge network. Since many of the offloaded tasks may require processing the same data items, the task allocation algorithms can exploit this to reduce the traffic in the networks and the number of edge servers needed to execute the tasks. Therefore, in this paper we design online algorithms for task allocation in edge computing systems that take into account the sharing of data among the tasks offloaded to the same server. We perform an extensive performance analysis by comparing our proposed algorithms with several sharing-oblivious baseline algorithms. The results show that our algorithms are able to reduce the amount of data transferred in the network by 30.2% to 92.8% and the number of utilized servers by 1% to 82.8% compared to the sharing-oblivious baseline algorithms.**

***Index Terms*—Edge computing, task allocation, online algorithms, data sharing.**

I. Introduction

Edge computing is a new computing paradigm offering real-time processing and analysis of data closer to the data sources, such as IoT devices, sensors, or end-user devices. This results in reducing latency, bandwidth usage, and faster response times leading to improved overall system efficiency. However, as the adoption of edge computing systems is growing, dealing with increasing amounts of data transferred in the edge network is a significant challenge. One of the main factors that edge providers need to consider to obtain an efficient deployment on edge servers, is exploiting the sharing of data among tasks that can be offloaded to the same server. For example, the same camera frame can be used by a task for face recognition and by another task for object detection [1]. A single audio recording might be analyzed by one task for voice recognition purposes and by another for detecting background noise levels [2]. An individual's GPS location data could be used by one task to provide real-time navigation and by another to offer location-based advertisements [3]. A video stream from a public place could be utilized by one task for monitoring the traffic flow and by another for assessing the crowd density [2]. Environmental sensor data, like air quality readings, might be used by one task to trigger pollution alerts and by another for long-term climate research [4]. Designing efficient algorithms for task allocation that consider data sharing among tasks is essential to improving the performance of the edge systems, which generally have limited computational and memory capacities.

In this paper, we take into account the sharing of data among tasks that can be offloaded on the same edge server and formulate the problem of *Online Data Sharing-Aware Task Allocation* (ODSTA). Managing the sharing of data items among many offloaded tasks on servers, is expected to reduce the total amount of data that needs to be stored at each server, and the traffic in the edge network that is associated with data transfers. We design three online algorithms for solving the ODSTA problem in edge computing systems. We consider an edge computing system composed of a set of heterogeneous servers, where the servers have different capacities for their computational resources (i.e., CPU and memory). Although our online algorithms are inspired by classical *sharing-oblivious* bin packing algorithms such as first-fit, best-fit and worst-fit, they differ from those in that when making allocation decisions they take into account the sharing among the data items stored in the main memory of edge servers.

This paper makes the following *contributions*:

- To the best of our knowledge, our work is the first to consider the design of *online* algorithms for task allocation in edge computing systems that take into account the sharing of data among tasks offloaded to the same server. Taking into account the sharing of data among tasks reduces the traffic in the edge network, the memory usage at the edge servers, and the number of servers needed to execute the tasks, thus, leading to an improved overall performance of the edge system.
- We develop a novel analytical model for capturing the sharing among online tasks and use it in the design of our algorithms.
- We design three online data sharing-aware algorithms (DSA) for task allocation in edge computing systems: DSA First-Fit (DSA-FF), DSA Best-Fit (DSA-BF), and DSA Worst-Fit (DSA-WF).
- We define an efficiency metric for DSA-BF and DSA-WF that characterizes the scarcity of resources at each server and it is used to make the allocation decisions.
- We perform an extensive performance analysis by comparing our proposed algorithms with several sharing-oblivious baseline online algorithms. The baseline algo-

rithms are the best existing online algorithms for task allocation that have as the objective minimizing the number of servers used to execute the tasks. The results show that our algorithms are able to reduce the amount of data transferred in the network by 30.2% to 92.8% and the number of utilized servers by 1% to 82.8% compared to the sharing-oblivious baseline algorithms.

Related work. In recent years, task allocation in edge computing has been the focus of extensive research [5]–[7]. Due to space limitations we will discuss only the research on task/resource allocation that take into account the sharing of data. Luo et al. [5] proposed a software-defined cooperative data sharing architecture and designed a cooperative data sharing scheduling algorithm to enable efficient cooperation between 5G and VANETs, and cooperation between communication and computing resources. Yang et al. [8] proposed energy consumption models of different types of equipment in Mobile Edge Computing and studied the placement of cloudlets on the edge nodes. Chu et al. [7] investigated adaptive data sharing techniques based on data flow analysis and bidirectional transformations and designed a hybrid offloading mechanism for allocating computations among agents and the cloud. Siew et al. [9] introduced a sharing economy-inspired business model to facilitate the sharing of excess resource quota among users, which led to a more efficient usage of resources. None of these works considered the task allocation with the objective of minimizing the number of utilized servers and the traffic in the network by taking into account the sharing of data among tasks.

The problem investigated in this paper is a variant of vector bin-packing [10] in which the items can have overlaps (have shared portions). This deviates from traditional formulations of bin-packing by considering the scenario where combining two or more items results in a reduced overall volume compared to the sum of their individual sizes. The online variant of the problem was introduced by Rampersaud and Grosu [11], [12] in the context of virtual machine (VM) packing in cloud computing systems. In their formulation, VM instances co-located on the same server can share some of their memory. They designed a family of sharing-aware online algorithms for solving the VM packing problem that take into account the sharing of memory among collocated VMs. Their algorithms lead to a reduction in the consumption of cloud resources, thereby enhancing efficiency in meeting users' demands. The problem considered in [11], [12] is different from the problem we consider in this paper. In this paper, we consider an edge computing system instead of a cloud computing system, tasks instead of VMs, and data items instead of pages in memory. Here, the goal is to minimize the number of servers in the edge system and the traffic in the network, while in the cloud setting the goal is to minimize the number of servers and the amount of memory allocated to the user's requested VMs.

In our previous work [13], we designed an offline data-sharing aware task allocation greedy algorithm for edge computing systems that maximizes the profit obtained from executing the tasks, while minimizing the traffic in the edge network. The problem considered in our previous work is an offline problem and did not consider that the tasks arrive online. Furthermore the objective is different, that is, minimizing the number of servers used instead of maximizing the profit.

TABLE I: Notation

Expression	Description
$\mathcal{S}$	Set of available servers.
$\mathcal{T}$	Set of tasks.
N	Number of tasks; $\|\mathcal{T}\| = N$.
M	Number of servers in edge system; $\|\mathcal{S}\| = M$.
A	Task-data matrix.
$\mathcal{D}$	Set of all data items associated with the tasks.
D_l	Data item l.
d_l	Size of data item l.
T_j	Task j.
r_j^u	CPU request by T_j (in MIPS).
t_j^a	Arrival time of task T_j.
t_j^e	Duration of task T_j.
S_k	Server k.
$\mathcal{S}_k$	Set of tasks allocated to server S_k.
C_k^u	CPU capacity of server S_k (in MIPS).
C_k^m	Memory capacity of server S_k.
Δ^k	Data allocation vector of server S_k
δ_j^k	Amount of shared data among T_j and tasks currently hosted by S_k.

Organization. The rest of the paper is organized as follows. Section II formulates the online data sharing-aware task allocation problem. Section III describes the proposed online algorithms. Section IV presents the experimental results. Section V concludes the paper and describes our future work.

II. Online Data Sharing-Aware Task Allocation Problem

In this section, we introduce the *Online Data Sharing-aware Task Allocation* (ODSTA) problem. We consider that the edge computing system consists of a set $\mathcal{S} = \{S_1, S_2, \ldots, S_M\}$ of M distributed edge servers, where each server $S_k \in \mathcal{S}$ is characterized by a tuple $[C_k^m, C_k^u]$, where C_k^m is its memory capacity, and C_k^u is its CPU capacity in GIPS (billion instructions per second). These edge servers host user's tasks from a set $\mathcal{T} = \{T_1, T_2, \ldots, T_N\}$ of N tasks, where $\mathcal{T}$ has an associated set of D data items, $\mathcal{D} = \{D_1, D_2, \ldots, D_D\}$, that are needed to execute the tasks. We denote the size of data item D_l by d_l (in GB), where $l = 1, 2, \ldots, D$. Each task T_j is characterized by a tuple $(r_j^u, t_j^a, t_j^e, [A]_{j,*})$, where r_j^u is T_j's CPU request in GIPS, t_j^a is T_j's arrival time, t_j^e is T_j's duration, and $[A]_{j,*}$ is the j-th row of the task-data matrix, A, corresponding to T_j. The *task-data matrix* A is a $N \times D$ matrix specifying for each task the data items needed by the task and their sizes, that is, $a_{jl} = d_l$, if task T_j requires data item D_l, and 0, otherwise.

The *objective* of the ODSTA problem is to find an allocation of users' tasks to servers such that the traffic in the edge network induced by the transfer of data to servers, and the number of servers used to execute the tasks are minimized. This is achieved by considering the data shared by the tasks when making the allocation decisions. For example, if three

tasks use the same data items and they are allocated (offloaded) to the same sever, the needed data items have to be transferred from users to this server only once. This is in contrast with the case in which the tasks are allocated to three different servers and the data items have to be transferred to three servers. This will increase the traffic in the network, thus increasing the latency of the tasks. Therefore, considering the data shared by the tasks when making the allocation decisions, we can reduce the total size of data that needs to be transferred in the network, and consequently, reduce the memory used at the servers. Minimizing the number of servers used is also important, leading to energy savings. The assignment of tasks to servers is online and there is no prior knowledge about the upcoming tasks.

To characterize the sharing of data for the upcoming task T_j hosted on server S_k, we define δ_j^k as the amount of data (memory) shared among the upcoming task T_j and all the tasks already assigned to server S_k, that is,

$$\delta_j^k = \sum_{l=1}^{D} a_{jl} \cdot [\![a_{jl} = \Delta_l^k]\!] \tag{1}$$

where $[\![C]\!]$ is the Iverson's bracket and Δ_l^k is entry l of *data allocation vector* Δ^k specifying the data items currently allocated to server S_k. The Iverson bracket $[\![C]\!]$ evaluates to 1 if the condition C is true, and 0, otherwise. The data allocation vector Δ^k has size D and is defined recursively as follows. If no task is allocated to S_k then $\Delta^k = [0]_D$. Let Δ^k be the vector before task T_j is allocated to server S_k and $\tilde{\Delta}^k$ the vector after T_j is allocated to S_k. Then, $\tilde{\Delta}^k = \Delta^k \boxplus [A]_{j,*}$, where $\boxplus$ is defined as follows,

$$\Delta^k \boxplus [A]_{j,*} = \begin{cases} \Delta_l^k & \text{if } a_{jl} = 0 \\ a_{jl} & \text{if } a_{jl} \neq 0 \end{cases} \tag{2}$$

Let Δ^k be the vector before task T_j is deallocated from server S_k and $\hat{\Delta}^k$ the vector after T_j is deallocated from S_k. Then, $\hat{\Delta}^k = \Delta^k \boxminus [A]_{j,*}$, where $\boxminus$ is defined as follows,

$$\Delta^k \boxminus [A]_{j,*} = \begin{cases} \Delta_l^k & \text{if } (\Delta_i^k = a_{jl}) \wedge (\exists T_i \in \mathcal{S}_k, i \neq j, a_{il} \neq 0) \\ 0 & \text{if } (\Delta_i^k = a_{jl}) \wedge (\forall T_i \in \mathcal{S}_k, i \neq j, a_{il} = 0) \\ \Delta_l^k & \text{if } a_{jl} = 0 \end{cases} \tag{3}$$

and $\mathcal{S}_k$ is the set of tasks allocated to server S_k.

A task T_j can be assigned to a server S_k if the following constraints are satisfied:

$$\tilde{C}_k^m - \sum_{l=1}^{D} a_{jl} + \delta_j^k \geq 0, \quad \forall T_j \in \mathcal{T}, \forall S_k \in \mathcal{S} \tag{4}$$

$$\tilde{C}_k^u - r_j^u \geq 0, \quad \forall T_j \in \mathcal{T}, \forall S_k \in \mathcal{S} \tag{5}$$

where $\tilde{C}_k^m$ and $\tilde{C}_k^u$ are the available memory and CPU capacities at server S_k. If there is no task assigned to server S_k, $\tilde{C}_k^m = C_k^m$ and $\tilde{C}_k^u = C_k^u$. Equation (4) is the memory capacity constraint for server S_k, guaranteeing that the available amount of memory of server S_k is not exceeded by allocating the memory needed for task T_j. Equation (5) is the CPU capacity constraint, guaranteeing that the CPU capacity of server S_k is not exceeded by allocating task T_j.

Example of ODSTA instance. In the following, we give an example of an ODSTA instance consisting of a set of six servers $\mathcal{S} = \{S_k | k = 1, \ldots, 6\}$ and a set of six tasks $\mathcal{T} = \{T_j | j = 1, \ldots, 6\}$ that need data items from a set $\mathcal{D} = \{D_l | l = 1, \ldots, 6\}$. The amount of CPU request r_j^u, the arrival time t_j^a, the duration t_j^e for each task, and the CPU and memory capacities (C_k^u and C_k^m) of the servers are given in Table II.

TABLE II: Example of ODSTA instance

Task	T_1	T_2	T_3	T_4	T_5	T_6
r_j^u (GIPS)	6	6	5	8	10	5
t_j^a (sec)	10	11	12	13	14	15
t_j^e (sec)	100	80	70	30	50	20
Server	S_1	S_2	S_3	S_4	S_5	S_6
C_k^m (GB)	25	21	18	32	35	34
C_k^u (GIPS)	20	10	12	14	16	13

The task-data matrix A that characterizes the sharing of data items among the tasks is:

$$\begin{array}{c} \\ T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \end{array} \begin{array}{c} \begin{array}{cccccc} D_1 & D_2 & D_3 & D_4 & D_5 & D_6 \end{array} \\ \begin{bmatrix} 2 & 4 & 0 & 5 & 0 & 1 \\ 2 & 0 & 6 & 5 & 0 & 0 \\ 2 & 4 & 0 & 0 & 0 & 1 \\ 2 & 4 & 6 & 5 & 4 & 0 \\ 2 & 4 & 6 & 5 & 4 & 1 \\ 0 & 0 & 0 & 5 & 4 & 1 \end{bmatrix} \end{array} \tag{6}$$

To illustrate how the above modeling of data sharing (Equations (1) to (4)) works, suppose that three tasks T_1, T_2, T_3 arrive one after another at the given arrival times, and are allocated to server S_1. T_1 is the first task to arrive and requires the data items D_1, D_2, D_4, and D_6 (i.e., $[A]_{1,*} = [2, 4, 0, 5, 0, 1]$). Initially, $\Delta^1 = [0]_6$ and since there is no task assigned to server S_1 yet, we have $\delta_1^1 = 0$. Thus, the total amount of data on server S_1 is $a_{11} + a_{12} + a_{14} + a_{16} = 12$ GB, and $\tilde{C}_1^m = 25 - 12 = 13$ GB. After the assignment of task T_1 on S_1, $\Delta^1 = [0, 0, 0, 0, 0, 0] \boxplus [2, 4, 0, 5, 0, 1] = [2, 4, 0, 5, 0, 1]$. The next incoming task is T_2 with $[A]_{2,*} = [2, 0, 6, 5, 0, 0]$. Both T_1 and T_2 require D_1 and D_4, and according to Equation (1), $\delta_2^1 = 2+5 = 7$ GB for task T_2. Thus, the total amount of data on server S_1 is $(2+4+5+1)+(2+6+5)-(2+5) = 18$ GB and $\Delta^1 = [2, 4, 0, 5, 0, 1] \boxplus [2, 0, 6, 5, 0, 0] = [2, 4, 6, 5, 0, 1]$. Next, T_3 is allocated to S_1 which has $[A]_{3,*} = [2, 4, 0, 0, 0, 1]$. For δ_3^1 we determine the amount of shared data among T_3, T_1, and T_2. Thus, according to Equation (1), $\delta_3^1 = 2 + 4 + 1 = 7$ GB. After the assignment of T_3 to S_1, $\Delta^1 = [2, 4, 6, 5, 0, 1] \boxplus [2, 4, 0, 0, 0, 1] = [2, 4, 6, 5, 0, 1]$, and the total amount of data for $\{T_1, T_2, T_3\}$ on server S_1 is $(2 + 4 + 6 + 5 + 1) + (2 + 4 + 1) - (2 + 4 + 1) = 18$ GB. If we consider a sharing-oblivious scenario, the total amount of data on S_1 would be $(2+4+5+1)+(2+6+5)+(2+4+1) = 32$ GB, which is larger than in the case of considering the data sharing at S_1.

III. Online Algorithms for ODSTA

In this section, we design online data sharing-aware task allocation algorithms that solve the ODSTA problem. The main objective of the algorithms for ODSTA is allocating online tasks to the servers while considering their data sharing (memory sharing), minimizing the number of servers used and the total amount of data transferred in the network. The design of our online algorithms is inspired by the classical online bin packing algorithms, First-Fit, Best-Fit, and Worst-Fit.

A. Data Sharing-Aware First Fit (DSA-FF) Algorithm

DSA-FF, given in Algorithm 1, is executed when a task arrives (lines 1-11), or a task completes its execution on the allocated server (lines 12-16). At the arrival of a task T_j, DSA-FF starts from the first server (i.e., $k = 1$, in line 2) and tries to find a server that has enough CPU and memory capacity to host T_j. In line 4, it calculates the amount of shared data between T_j and the currently allocated tasks on server S_k according to Equation (1). Then, it checks if the current server S_k has enough CPU and memory capacity to host the incoming task (line 5). If there are enough resources on the current server, it updates the data allocation vector Δ^k (line 6) according to Equation (2), allocates task T_j to the current server S_k (line 7), and updates the CPU and memory capacities of server S_k (line 8). Then, it exits the while loop (line 9). Otherwise, it increments k (line 11), which means that if the current server S_k does not have enough resources to host T_j, then it checks the next available server. After the completion of task T_j on server S_k, DSA-FF removes T_j from set $\mathcal{S}_k$ of tasks allocated to S_k (line 13) and updates the data allocation vector according to Equation (3) (line 14). It also updates δ_j^k considering the updated data allocation vector Δ^k (line 15), and finally frees up server S_k's resources (line 16).

DSA-FF: Illustrative example. To illustrate how DSA-FF works, we consider the example of ODSTA instance described in Section II. Figure 1, shows the allocation of the tasks to servers at the arrival of each of the six tasks. At T_1's arrival ($t_1^a = 10$), DSA-FF checks if the first server, S_1, can host T_1. Since there is no task allocated to S_1, and S_1 has enough CPU and memory, T_1 is allocated to it. T_2 arrives at $t_2^a = 11$, and since $[\tilde{C}_1^m, \tilde{C}_1^u] - [(a_{21} + a_{23} + a_{24}) - \delta_2^1, r_2^u] = [13, 14] - [(2 + 6 + 5) - (2 + 5), 6] = [7, 8] \geq [0, 0]$, S_1 can host T_2 as well. For the allocation of T_2 on server S_1 we consider the data sharing between T_2 and previously allocated task T_1 on server S_1. Since T_1 and T_2 have D_1 and D_4 in common, $\delta_2^1 = 2 + 5 = 7$ GB. Due to space limitations we do not describe in detail how the allocation of T_3, T_4, T_5 and T_6 is decided, but in Figure 1 we show the state of the allocation of these tasks to servers. The tasks will complete their execution according to the order given by the sum of their arrival time and duration (i.e., $T_6, T_4, T_5, T_3, T_2, T_1$) and DSA-FF will release the allocated resources. The total number of active servers utilized by DSA-FF for the ODSTA instance considered here is 4, and the total amount of data transferred in the edge network is $(2+4+6+5+1)+(2+4+6+5+4)+(5+4+1)+(2+4+6+5+4+1) = 71$ GB. If we consider the sharing-oblivious case (standard first-fit algorithm), the total amount of data transferred in the network is 85 GB and the number of utilized servers is 4.

Algorithm 1 Data Sharing-Aware First Fit (DSA-FF)

1: **at** task T_j arrival **do**
2: $k \leftarrow 1$
3: **while** $(k \leq |\mathcal{S}|))$ **do**
4: $\delta_j^k \leftarrow \sum_{l=1}^{D} a_{jl} \cdot [\![a_{jl} = \Delta_l^k]\!]$
5: **if** $([\tilde{C}_k^m, \tilde{C}_k^u] - [\sum_{l=1}^{N} a_{jl} - \delta_j^k, r_j^u] \geq [0, 0])$ **then**
6: $\Delta^k \leftarrow \Delta^k \boxplus [A]_{j,*}$
7: $\mathcal{S}_k \leftarrow \mathcal{S}_k \cup \{T_j\}$
8: $[\tilde{C}_k^m, \tilde{C}_k^u] \leftarrow ([\tilde{C}_k^m, \tilde{C}_k^u] - [\sum_{l=1}^{N} a_{jl} - \delta_j^k, r_j^u])$
9: **break**
10: **else**
11: $k \leftarrow k + 1$
12: **at** task T_j completion on server S_k **do**
13: $\mathcal{S}_k \leftarrow \mathcal{S}_k \backslash \{T_j\}$
14: $\Delta^k \leftarrow \Delta^k \boxminus [A]_{j,*}$
15: $\delta_j^k \leftarrow \sum_{l=1}^{D} a_{jl} \cdot [\![a_{jl} = \Delta_l^k]\!]$
16: $[\tilde{C}_k^m, \tilde{C}_k^u] \leftarrow ([\tilde{C}_k^m, \tilde{C}_k^u] + [\sum_{l=1}^{N} a_{jl} - \delta_j^k, r_j^u])$

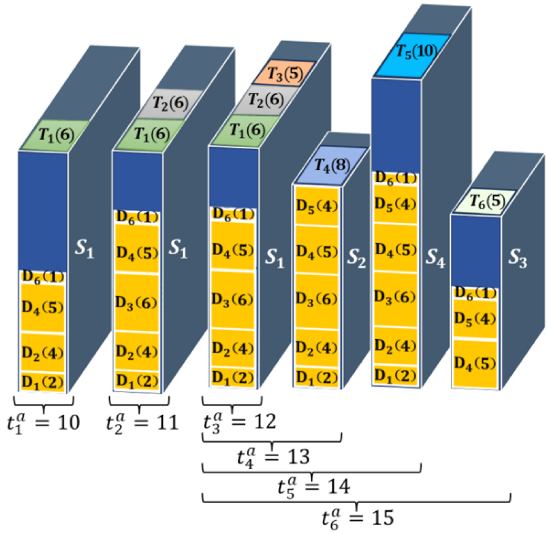


Fig. 1: DSA-FF execution example (the vertical dimension corresponds to the memory, while the depth dimension corresponds to the CPU; the numbers in parentheses are the size of CPU requests and the size of data items).

Time complexity. The worst case time complexity of DSA-FF at a task arrival is $\mathcal{O}(MD)$. This is because in the worst case it has to check the capacity of M servers and for each server it has to go over D entries of the data allocation vector to determine the shared items. The worst case time complexity at the completion of a task is $\mathcal{O}(D)$, and is due to the update of the data allocation vector.

B. Data Sharing-Aware Best Fit (DSA-BF) Algorithm

To design DSA-BF, we define an *efficiency metric* that characterizes the scarcity of resources at each server. The classical best fit bin packing algorithm attempts to place

each new item into the bin with the maximum load (smallest available space) in which it fits. Thus, if such a bin is found, the new item is placed inside it. To find the maximum load in the case of our edge system, where each server is characterized by two main parameters, CPU capacity and memory size, we define the efficiency metric E_j^k as follows,

$$E_j^k = \begin{cases} \left[\alpha(\tilde{C}_k^u - r_j) + \beta(\tilde{C}_k^m - (\sum_{l=1}^{N} a_{jl} - \delta_j^k))\right]^{-\frac{1}{2}} \\ \quad \text{if } (\tilde{C}_k^u - r_j \geq 0) \text{ and } (\tilde{C}_k^m - (\sum_{l=1}^{N} a_{jl} - \delta_j^k) \geq 0) \\ 0 \quad \text{otherwise} \end{cases} \tag{7}$$

where α and β are the weight parameters for CPU capacity and memory size. The weights are used to tune the algorithm and can be adjusted to prioritize one aspect over the other when allocating resources on a server, for example giving more importance to CPU or memory when making the allocation decisions. If α is greater than β, then the CPU capacity will play a more significant role in the efficiency metric. Increasing α would mean that servers with more spare CPU capacity will be considered more efficient, and have better chance to be allocated. If β is greater than α, then the memory size will be more influential in the allocation decision. Thus, servers with more memory available will be deemed more efficient. The efficiency is only calculated when the remaining CPU and memory capacity (after accounting for demands) are both positive. If either is negative, meaning that the server cannot accommodate the new demands, the efficiency is set to zero, indicating that the server should not be considered for receiving additional load.

In practice, α and β are likely to be determined through a combination of business requirements, performance metrics, and cost considerations. Different types of edge applications and services have varying resource needs. For example, image processing and virtual reality applications may require more CPU cycles to perform calculations efficiently, whereas applications like databases or caching services may demand more memory to store and quickly access large amounts of data. By adjusting α and β, the algorithm can be tailored to favor servers that offer the required resources more abundantly. Sometimes, in an edge computing system, one type of resource may be more scarce or expensive than the other. If CPU is a more scarce resource, a higher α would ensure that tasks are directed to servers with sufficient CPU availability. Conversely, if memory is scarce, β would be increased to safeguard memory resources. Moreover, different resources also come with different costs. If CPU capacity is more expensive than memory, a higher α value could be used to optimize for cost by ensuring that CPU resources are utilized as efficiently as possible, potentially saving money for the edge provider. For some services, performance may be critically dependent on one resource over the other. If a server's performance is more sensitive to CPU than memory, one would increase α to reduce latency or increase throughput by allocating tasks to servers where they are less likely to be CPU-constrained. Parameters α and β can be adjusted to achieve a balanced load across a system with multiple servers. For instance, if servers tend to run out of memory before running out of CPU, increasing β would help balance the usage of memory across the servers, potentially leading to a more uniform usage of both types of resources.

Algorithm 2 Data Sharing-Aware Best Fit (DSA-BF)

1: **at** task T_j arrival **do**
2: **for** $k = 1, \ldots, |\mathcal{S}|$ **do**
3: $\delta_j^k \leftarrow \sum_{l=1}^{D} a_{jl} \cdot [\![a_{jl} = \Delta_l^k]\!]$
4: **if** $([\tilde{C}_k^m, \tilde{C}_k^u] - [\sum_{l=1}^{N} a_{jl} - \delta_j^k, r_j^u] \geq [0, 0])$ **then**
5: $E_j^k \leftarrow \left[\alpha(\tilde{C}_k^u - r_j) + \beta(\tilde{C}_k^m - (\sum_{l=1}^{N} a_{jl} - \delta_j^k))\right]^{-\frac{1}{2}}$
6: **else**
7: $E_j^k \leftarrow 0$
8: $\tilde{k} \leftarrow \text{argmax}_k\{E_j^k\}$
9: $\Delta^{\tilde{k}} \leftarrow \Delta^{\tilde{k}} \boxplus [A]_{j,*}$
10: $\mathcal{S}_{\tilde{k}} \leftarrow \mathcal{S}_{\tilde{k}} \cup \{T_j\}$
11: $[\tilde{C}_{\tilde{k}}^m, \tilde{C}_{\tilde{k}}^u] \leftarrow ([\tilde{C}_{\tilde{k}}^m, \tilde{C}_{\tilde{k}}^u] - [\sum_{l=1}^{N} a_{jl} - \delta_j^{\tilde{k}}, r_j^u])$
12: **at** task T_j completion on server S_k **do**
13: $\mathcal{S}_k \leftarrow \mathcal{S}_k \backslash \{T_j\}$
14: $\Delta^k \leftarrow \Delta^k \boxminus [A]_{j,*}$
15: $\delta_j^k \leftarrow \sum_{l=1}^{D} a_{jl} \cdot [\![a_{jl} = \Delta_l^k]\!]$
16: $[\tilde{C}_k^m, \tilde{C}_k^u] \leftarrow ([\tilde{C}_k^m, \tilde{C}_k^u] + [\sum_{l=1}^{N} a_{jl} - \delta_j^k, r_j^u])$

Finding the server with the maximum load is equivalent to finding the server with the minimum of available resource capacity. Thus, in E_j^k we consider the available capacity for CPU and memory i.e., $\tilde{C}_k^u - r_j$ and $\tilde{C}_k^m - (\sum_{l=1}^{N} a_{jl} - \delta_j^k)$, respectively. Since the square root of the sum of these values is in the denominator of the efficiency metric, we will consider the maximum of E_j^k among the servers. It is important to mention that the efficiency metric E_j^k takes into account the sharing of memory among tasks that are offloaded to the same server S_k.

DSA-BF, given in Algorithm 2, is executed when a task arrives (lines 1-11), or a task completes its execution on the allocated server (lines 12-16). Upon arrival of task T_j, for each server S_k in the system, it determines δ_j^k (line 3), and if there is enough CPU and memory capacity at S_k then it calculates the efficiency metric using Equation (7), otherwise it sets E_j^k to 0 (lines 4-7). In line 8, DSA-BF picks the server $S_{\tilde{k}}$ that has the maximum efficiency metric. In lines 9-11, it updates the data allocation vector $\Delta^{\tilde{k}}$ for the selected server $S_{\tilde{k}}$ according to Equation (2), allocates T_j on server $S_{\tilde{k}}$, and updates its CPU and memory capacities. Upon completion of task T_j on server S_k, DSA-BF removes T_j from set $\mathcal{S}_k$ of tasks allocated to S_k (line 13) and updates the data allocation vector (line 14). It also updates the amount of shared data among currently allocated tasks on S_k (line 15), and releases T_j's allocated resources (line 16).

DSA-BF: Illustrative example. To illustrate how DSA-BF works, we consider the example of ODSTA instance described in Section II. Figure 2, shows the allocation of the tasks to servers at the arrival of each of the six tasks. At T_1's arrival ($t_1^a = 10$), DSA-BF calculates δ_j^k for all servers and since all servers have enough capacity it calcu-

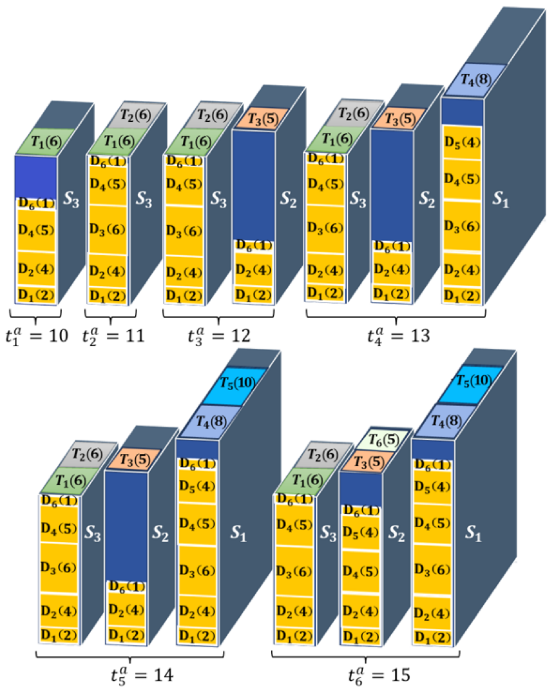


Fig. 2: DSA-BF execution example (the vertical dimension corresponds to the memory while the depth dimension corresponds to the CPU; the numbers in parentheses are the size of CPU requests and the size of data items).

lates their efficiency metric (with $\alpha = \beta = 0.5$) (line 5). Thus, $E_1^3 = [0.5(12-6)+0.5(18-12)]^{-1/2} = 0.408$ is the maximum value of efficiency metric among all servers. Therefore, DSA-BF selects server S_3, which is the server with minimum available capacity, to host T_1. It then updates $[\tilde{C}_3^m, \tilde{C}_3^u] - [(a_{11}+a_{12}+a_{14}+a_{16}) - \delta_1^3, r_1^u] = [18,12] - [(2+4+5+1) - 0, 6] = [6,6] \geq [0,0]$. At T_2's arrival ($t_2^a = 11$), S_3 is determined to have the maximum value of efficiency metric, thus, T_2 is allocated to S_3 as well and $[\tilde{C}_3^m, \tilde{C}_3^u] - [(a_{21}+a_{23}+a_{24}) - \delta_2^3, r_2^u] = [6,6] - [(2+6+5)-(2+5), 6] = [0,0]$. The next task to arrive is T_3 ($t_3^a = 12$). Since S_3 does not have enough capacity to host T_3, $E_3^3 = 0$. S_2 is the server with the maximum value of the efficiency metric and it can host T_3, and we have $[\tilde{C}_2^m, \tilde{C}_2^u] - [(a_{31}+a_{32}+a_{36}) - \delta_3^2, r_3^u] = [21,10] - [(2+4+1)-0, 5] = [14,5] \geq [0,0]$. The next task to arrive is T_4 ($t_4^a = 13$). Since both S_3 and S_2 do not have enough capacity to host T_4, DSA-BF sets their efficiency metric to 0. S_1 is the server with the maximum value of the efficiency metric and it can host T_4, and we have $[\tilde{C}_1^m, \tilde{C}_1^u] - [(a_{41}+a_{42}+a_{43}+a_{44}+a_{45}) - \delta_4^1, r_4^u] = [25,20] - [(2+4+6+5+4) - 0, 8] = [4,12] \geq [0,0]$. T_5 is allocated to S_1 and $[\tilde{C}_1^m, \tilde{C}_1^u] - [(a_{51}+a_{52}+a_{53}+a_{54}+a_{55}+a_{56}) - \delta_5^1, r_5^u] = [4,12] - [(2+4+6+5+4+1)-(2+4+6+5+4), 10] = [3,2]$. The last task to arrive is T_6 ($t_6^a = 15$) which is allocated to S_2 because E_6^2 is the maximum efficiency metric. Thus, $[\tilde{C}_2^m, \tilde{C}_2^u] - [(a_{64}+a_{65}+a_{66}) - \delta_6^2, r_6^u] = [14,5] - [(5+4+1)-1, 5] = [5,0] \geq [0,0]$. The tasks will complete their execution according to the order given by the sum of their arrival time and duration and DSA-BF will release the allocated resources. The total number of active servers utilized by DSA-BF for the ODSTA instance considered here is 3, and the total amount of data transferred in the edge network is $(2+4+6+5+4+1)+(2+4+5+4+1)+(2+4+6+5+1) = 56$ GB. If we consider the sharing-oblivious case (standard best-fit algorithm), the total amount of data transferred in the network is 85 GB and the number of utilized servers is 5.

Time complexity. The worst case time complexity of DSA-BF at a task arrival is $\mathcal{O}(MD)$. This is because in the worst case it has to check the capacity and efficiency metric for M servers, and for each server it has to go over D entries of the data allocation vector to determine the shared items. The worst case time complexity at the completion of a task is $\mathcal{O}(D)$, and is due to the update of the data allocation vector.

C. Data Sharing-Aware Worst Fit (DSA-WF) Algorithm

To design DSA-WF we use the same efficiency metric as in the case of DSA-BF (Equation (7)) to characterize the scarcity of servers' resources. The classical worst fit bin packing algorithm attempts to place each new item into the bin with the minimum load (largest available space) in which it fits. Thus, if such a bin is found, the new item is placed inside it. Thus, to find the worst fit server DSA-WF uses the efficiency metric. DSA-WF, given in Algorithm 3, has the same structure as DSA-BF, the difference is that DSA-WF determines the server that gives the minimum efficiency metric instead of the one that gives the maximum (line 8 of both algorithms).

DSA-WF: Illustrative example. To illustrate how DSA-WF works, we consider the example of ODSTA instance described in Section II. Figure 3, shows the allocation of the tasks to servers at the arrival of each of the six tasks. At T_1's arrival ($t_1^a = 10$), DSA-WF calculates δ_j^k for all servers and since all servers have enough capacity it calculates their efficiency metric (with $\alpha = \beta = 0.5$) (line 5).

Algorithm 3 Data Sharing-Aware Worst Fit (DSA-WF)

1: **at** task T_j arrival **do**
2: **for** $k = 1, \ldots, |\mathcal{S}|$ **do**
3: $\delta_j^k \leftarrow \sum_{l=1}^{D} a_{jl} \cdot [\![a_{jl} = \Delta_l^k]\!]$
4: **if** $([\tilde{C}_k^m, \tilde{C}_k^u] - [\sum_{l=1}^{N} a_{jl} - \delta_j^k, r_j^u] \geq [0,0])$ **then**
5: $E_j^k \leftarrow \left[\alpha(\tilde{C}_k^u - r_j) + \beta(\tilde{C}_k^m - (\sum_{l=1}^{N} a_{jl} - \delta_j^k))\right]^{-\frac{1}{2}}$
6: **else**
7: $E_j^k \leftarrow 0$
8: $\tilde{k} \leftarrow \text{argmin}_k \{E_j^k\}$
9: $\Delta^{\tilde{k}} \leftarrow \Delta^{\tilde{k}} \boxplus [A]_{j,*}$
10: $\mathcal{S}_{\tilde{k}} \leftarrow \mathcal{S}_{\tilde{k}} \cup \{T_j\}$
11: $[\tilde{C}_{\tilde{k}}^m, \tilde{C}_{\tilde{k}}^u] \leftarrow ([\tilde{C}_{\tilde{k}}^m, \tilde{C}_{\tilde{k}}^u] - [\sum_{l=1}^{N} a_{jl} - \delta_j^{\tilde{k}}, r_j^u])$
12: **at** task T_j completion on server S_k **do**
13: $\mathcal{S}_k \leftarrow \mathcal{S}_k \backslash \{T_j\}$
14: $\Delta^k \leftarrow \Delta^k \boxminus [A]_{j,*}$
15: $\delta_j^k \leftarrow \sum_{l=1}^{D} a_{jl} \cdot [\![a_{jl} = \Delta_l^k]\!]$
16: $[\tilde{C}_k^m, \tilde{C}_k^u] \leftarrow ([\tilde{C}_k^m, \tilde{C}_k^u] + [\sum_{l=1}^{N} a_{jl} - \delta_j^k, r_j^u])$

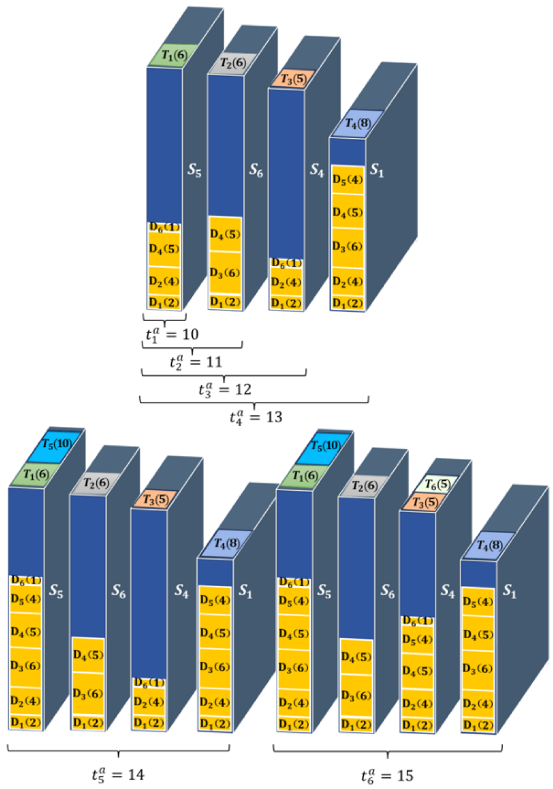


Fig. 3: DSA-WF execution example (the vertical dimension corresponds to the memory while the depth dimension corresponds to the CPU; the numbers in parentheses are the size of CPU requests and the size of data items).

Thus, $E_1^5 = [0.5(12-6)+0.5(18-12)]^{-1/2} = 0.24$ is the minimum value of efficiency metric among all servers. Therefore, DSA-WF selects server S_5, which is the server with maximum available capacity, to host T_1. It then updates $[\tilde{C}_5^m, \tilde{C}_5^u] - [(a_{11}+a_{12}+a_{14}+a_{16}) - \delta_1^5, r_1^u] = [35,16] - [(2+4+5+1)-0,6] = [23,10] \geq [0,0]$. At T_2's arrival ($t_2^a = 11$), S_6 is determined to have the minimum value of efficiency metric, thus, T_2 is allocated to S_6 and $[\tilde{C}_6^m, \tilde{C}_6^u] - [(a_{21} + a_{23} + a_{24}) - \delta_2^6, r_2^u] = [34,13] - [(2+6+5),6] = [21,7]$. The next task to arrive is T_3 ($t_3^a = 12$). Since S_4 has the minimum value of efficiency metric, thus, T_3 is allocated to S_4, and we have $[\tilde{C}_4^m, \tilde{C}_4^u] - [(a_{31}+a_{32}+a_{36}) - \delta_3^4, r_3^u] = [32,14] - [(2+4+1)-0,5] = [25,9] \geq [0,0]$. The next task to arrive is T_4 ($t_4^a = 13$) and S_1 has the minimum value of efficiency metric, thus, S_1 can host T_4, and we have $[\tilde{C}_1^m, \tilde{C}_1^u] - [(a_{41}+a_{42}+a_{43}+a_{44}+a_{45}) - \delta_4^1, r_4^u] = [25,20] - [(2+4+6+5+4)-0,8] = [4,12] \geq [0,0]$. T_5 is allocated to S_5 due to its efficiency's metric minimum value and $[\tilde{C}_5^m, \tilde{C}_5^u] - [(a_{51}+a_{52}+a_{53}+a_{54}+a_{55}+a_{56}) - \delta_5^5, r_5^u] = [23,10] - [(2+4+6+5+4+1)-(2+4+5+1),10] = [13,0]$. The last task to arrive is T_6 ($t_6^a = 15$) which is allocated to S_4 because E_6^4 is the minimum efficiency metric. Thus, $[\tilde{C}_4^m, \tilde{C}_4^u] - [(a_{64}+a_{65}+a_{66}) - \delta_6^2, r_6^u] = [25,9] - [(5+4+1)-1,5] = [16,4] \geq [0,0]$. The tasks will complete their execution according to the order given by the sum of their arrival time and duration and DSA-WF will release the allocated resources.

TABLE III: Distributions and parameters used for evaluation ($\mathcal{N}$ and $\mathcal{U}$ denote the normal distribution and the uniform distributions, respectively)

Parameter	Distribution/Value
Number of Servers	120 heterogeneous servers
CPU Capacities	$\mathcal{N}(\mu=280, \sigma=100)$, range $[200, 500]$ GIPS
Memory Capacities	$\mathcal{N}(\mu=16, \sigma=1.5)$, range $[15, 20]$ GB
Number of Tasks	500
Task's CPU requests size	low: $\mathcal{N}(\mu=15, \sigma=5)$, range $[1, 65]$ GIPS medium: $\mathcal{N}(\mu=35, \sigma=5)$, range $[1, 65]$ GIPS high: $\mathcal{N}(\mu=55, \sigma=5)$, range $[1, 65]$ GIPS
Task's Arrival Time	$\mathcal{U}[5, 50]$ seconds
Task's Duration	$\mathcal{U}[5, 50]$ seconds
Data Items	100
Data Size	$\mathcal{N}(\mu=25, \sigma=4)$, range $[1, 50]$ GB

The total number of active servers utilized by DSA-WF for the ODSTA instance considered here is 4, and the total amount of data transferred in the edge network is $(2+4+6+5+4)+(2+4+5+4+1)+(2+4+5+6+4+1)+(2+6+5) = 72$ GB. If we consider the sharing-oblivious case (standard best-fit algorithm), the total amount of data transferred in the network is 85 GB and the number of utilized servers is 4.

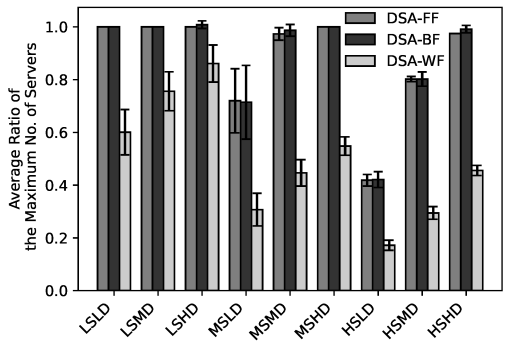
Time complexity. The worst case time complexity of DSA-WF at a task arrival is $\mathcal{O}(MD)$. The worst case time complexity at the completion of a task is $\mathcal{O}(D)$. The same reasoning as in the case of DSA-BF applies here.

IV. Experimental Analysis

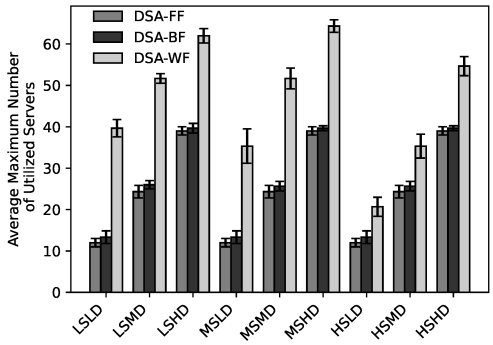
In this section, we investigate the performance of the proposed online data sharing-aware algorithms, and compare them with baseline sharing-oblivious online algorithms. Such sharing-oblivious algorithms (e.g. First-Fit, Best-Fit) have been employed as task allocation and as baseline benchmark algorithms in edge computing [14]–[17], and thus, we consider them as baselines in our performance analysis. We implement the algorithms in Java and run the simulation experiments on a system with 8 cores Intel i7-11700 at 2.50GHz, 16GB of memory, and 512 GB SSD. First, we describe the experimental setup and then analyze the results.

A. Experimental Setup

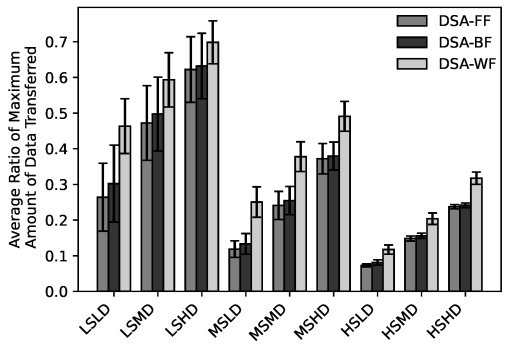
Servers' CPU and memory capacities. We consider an edge computing system composed of $M = 120$ heterogeneous servers and draw their CPU capacities from the normal distribution with mean $\mu = 280$ GIPS, standard deviation $\sigma = 100$, and range $[200, 500]$ GIPS. We draw the servers' memory capacities from the normal distribution with mean $\mu = 16$ GB, standard deviation $\sigma = 1.5$, and range $[15, 20]$ GB. The ranges of CPU and memory capacities were selected to match those of common systems (e.g., Intel i7 with 16GB of memory). Table III provides a summary of the simulation parameters and their distributions used in our experiments.



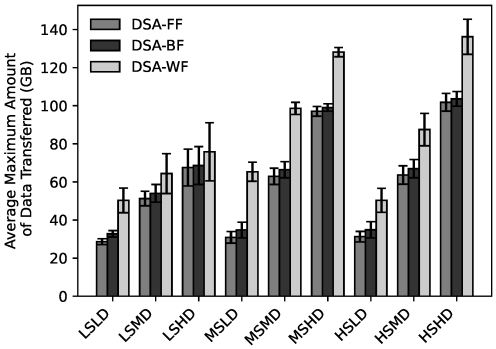
(a) Relative maximum number of utilized servers.



(b) Average maximum number of utilized servers.



(c) Relative maximum amount of data transferred.



(d) Average maximum amount of data transferred.

Fig. 4: Performance of the proposed data sharing-aware algorithms for the nine sharing and demand scenarios (here DSA-BF uses $\alpha = \beta = 0.5$).

Tasks' CPU requests. To characterize the relationship between server's total capacity (in GIPS) and the total amount of computational request, we define a demand ratio, ρ as follows:

$$\rho = \frac{\sum_{i=1}^{N} r_i}{\sum_{j=1}^{M} C_j}, \quad 0 \leq \rho \leq 1 \tag{8}$$

Based on Equation (8), we consider three different cases for computational requests by tasks:

(i) *Low Demand* (LD) ($\rho \ll 1$), when the total requested capacity is much smaller than the total available capacity on the edge servers;
(ii) *Medium Demand* (MD) ($\rho \approx 1/2$), when the total requested capacity is about half the total available capacity of the edge servers; and
(iii) *High Demand* (HD) ($\rho \approx 1$) when the total requested capacity is approximately equal to the total capacity available on the edge servers.

We consider $N = 500$ tasks. The tasks' CPU requests sizes are drawn from the normal distribution with standard deviation $\sigma = 5$ GIPS, range $[1, 65]$ GIPS, and mean $\mu = 15, 35$, and 55 GIPS for low, medium, and high demand, respectively. The tasks' arrival time and duration are drawn from the uniform distribution within the range $[5, 50]$ seconds.

Task-data matrix and data sharing. The data item's sizes are drawn from the normal distribution with mean $\mu = 25$, standard deviation $\sigma = 4$, and a range of $[1, 50]$ MB. We utilize the Erdös-Rényi random graph model [18] to create the bipartite graph that represents the data sharing pattern of the generated taskset. The adjacency matrix of this graph is used as the task-data matrix A. In the Erdös-Rényi model, 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 $p \in [0, 1]$, called the *sharing degree* here, characterizes the sharing among tasks. Larger values of p correspond 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:

(i) *Low Sharing* (LS) ($0 \leq p \leq 0.3$), where only a few tasks share data items;
(ii) *Medium Sharing* (MS) ($0.3 < p \leq 0.6$); and
(iii) *High Sharing* (HS) ($0.6 < p \leq 1$), where a large number of tasks share data items.

For our experimental results we consider task-matrix $A_{500\times100}$, representing a bipartite graph with 500 tasks and 100 data items.

B. Experimental Results

To analyze the performance of the proposed online data sharing-aware algorithms, we consider nine different cases

according to the three levels of sharing and demand, i.e., {LS, MS, HS} × {LD, MD, HD}. Each task is characterized by a certain size of its computational request and the data items needed, and each server has a certain CPU and memory capacities, as described in the previous section. We compare the performance of the proposed online data sharing-aware algorithms with that of the sharing-oblivious baseline algorithms, First-Fit (FF), Best-Fit (BF), and Worst-Fit (WF), in terms of the maximum number of utilized servers and the maximum amount of data transferred in the edge network. The results were obtained by executing the algorithms three times and computing the average values of the metrics described above and computing the standard deviation. The sharing-oblivious algorithms considered here are: First-Fit (FF), Best-Fit (BF), and Worst-Fit (WF).

Maximum number of utilized servers. For each of the online algorithms we recorded the number of utilized servers at each task arrival and termination and determine the maximum among those values. Then, we determine the ratio of the maximum number of utilized servers in the case of the data sharing-aware algorithms and that of the sharing-oblivious algorithms.

Figure 4a, shows this ratio for all nine types of instances with different levels of sharing and demand. The ratio decreases as the sharing of data items among the tasks increases. For example, in the case of LSLD, DSA-FF utilizes the same maximum number of servers as FF, corresponding to a ratio of 1, while in the case of HSLD the DSA-FF utilizes 59% less servers than FF (corresponding to a ratio of 0.41). DSA-BF obtains almost the same reduction in the maximum number of servers as that obtained by DSA-FF for those cases. The largest decrease in the maximum number of servers, of 82.8%, is obtained by DSA-WF in the case of HSLD. The decrease in the maximum number of utilized servers observed for instances with high amount of data sharing among tasks is due to the fact that the total memory required for the tasks allocated to the same server is reduced allowing the server to host more tasks, and thus reduce the number of servers needed to host the tasks. As the demand for CPU increases for the same level of sharing, we observe that the ratio increases, that is the reduction in the number of utilized servers decreases. This is due to the fact that the demand for CPU is high and sharing of data does not help much in allocating more tasks to servers. In other words, if tasks have a high CPU demand, the advantage of shared data is diminished. The servers quickly reach their CPU capacity, limiting the number of tasks they can handle, and thus, more servers are still needed despite the memory savings. This results in a smaller reduction in the number of servers used compared to scenarios where CPU demand is lower.

Figure 4b, shows the average of maximum number of utilized servers by the proposed data sharing-aware algorithms. We observe that DSA-FF obtains the best performance for all sharing and demand scenarios, followed very closely by DSA-BF. As the CPU demand increases, the number of utilized servers increases. This is due to the fact that more tasks need to be executed, which leads to the utilization of more servers. In addition, at higher CPU demands, the currently utilized servers may not have enough capacity to handle all the tasks if they are operating near their maximum capacities.

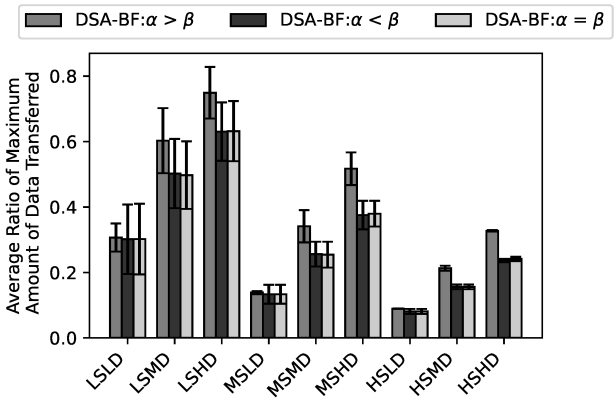


Fig. 5: The effect of efficiency metric weights (α, β) on DSA-BF's performance: Relative maximum amount of data transferred in the network.

Maximum amount of data transferred in the edge network. For each of the online algorithms we recorded the amount of data transferred in the edge network at each task arrival and termination and determine the maximum among those values. Then, we determine the ratio of the maximum amount of data transferred in the case of the proposed data sharing-aware algorithms and that of the sharing-oblivious algorithms. Figure 4c, shows this ratio for all nine types of instances with different levels of sharing and demand. The ratio decreases as the sharing of data items among the tasks increases. For example, in the case of LSLD, DSA-FF leads to 74% less data transferred in the network than FF (corresponding to a ratio of 0.26), while in the case of HSLD, DSA-FF leads to 92.8% less data transferred than FF. A similar behavior is exhibited by DSA-BF which obtains a little bit smaller reduction in the data transferred for those cases (about 70% for LSLD and 92% for HSLD). The decrease in the maximum amount of data transferred in the network observed for instances with high amount of data sharing among tasks is due to the fact that tasks share many items and if they are co-located on the same server those shared data items are transferred only once, thus, the more shared data items the less data transfers. As the demand for CPU increases for the same level of sharing, we observe that the ratio increases, that is, the reduction in the maximum amount of transferred data increases. This is due to the fact that the demand for CPU is high and tasks that share data items cannot be collocated on the same server.

Figure 4d shows the average amount of data transferred in the network by the proposed data sharing-aware algorithms. We observe that DSA-FF obtains the best performance for all sharing and demand scenarios, followed very closely by DSA-

BF. As the CPU demand increases, the amount of data that needs to be transferred also increases.

The effect of efficiency metric weights (α, β) on DSA-BF's performance. We investigate how the choice of the weights (α, β) used in the efficiency function influences the amount of data transferred in the edge network by DSA-BF. We consider three case: (i) $\alpha > \beta$, where $\alpha = 0.9998$, and $\beta = 0.0002$; (ii) $\alpha < \beta$, where $\alpha = 0.0002$, and $\beta = 0.9998$; and (iii) $\alpha = \beta$, where $\alpha = 0.5$, and $\beta = 0.5$. Figure 5, shows the average ratio of the maximum amount of data transferred by DSA-BF with respect to BF for these three cases. In all nine sharing and demand scenarios DSA-BF with $\alpha = 0.9998, \beta = 0.0002$ (i.e., case (i)) has the largest ratio among the three cases, that is, the amount of transferred data relative to BF is the largest. This is due to the fact, that the contribution of memory capacity term to the denominator of the efficiency metric is much smaller compared to that of the CPU capacity term, and thus, the CPU capacity term influences more the decisions. In case (ii) and (iii) the contribution of memory capacity term to the denominator of the efficiency metric is increased and thus, influences more the decisions, selecting servers with better fit for the memory requests. Therefore, DSA-BF is able to fit more items in the memory of the servers, leading to a reduction in the amount of data transferred in the edge network.

V. Conclusion and Future Work

We designed three online algorithms (DSA-FF, DSA-BF, and DSA-WF) for task allocation in edge computing systems that take into account the sharing of data among tasks offloaded to the same server. Taking into account the sharing of data among tasks reduces the traffic in the edge network, the memory usage at the edge servers, and the number of servers needed to execute the tasks, thus, leading to an improved overall performance of the edge system. We performed an extensive performance analysis by comparing our proposed algorithms with several online sharing-oblivious baseline algorithms. The results show that our algorithms are able to reduce the amount of data transferred in the network by 30.2% to 92.8% and the number of utilized servers by 1% to 82.8% compared to the sharing-oblivious baseline algorithms. In our future work, we plan to explore further optimizations of these algorithms to improve their performance and also determine their theoretical performance guarantees. We also plan to deploy our algorithms on an experimental edge computing testbed and investigate their performance.

Acknowledgment

This research was supported in part by the US National Science Foundation under Grants CCF-2118202 and CNS-2231523.

References

[1] H. Lee, G. Hong, and D. Shin, "Shareable camera framework for multiple computer vision applications," in *Proc. 20th Int. Conf. on Advanced Communication Technology*. IEEE, 2018, pp. 669–674.

[2] G. Dermler, T. Gutekunst, E. Ostrowski, and F. Ruge, "Sharing audio/video applications among heterogeneous platforms," in *Proc. 5th IEEE COMSOC Int. Workshop on Multimedia Communications*. IEEE, 1994, pp. 1–5.

[3] C. Li, Y. Fu, F. R. Yu, T. H. Luan, and Y. Zhang, "Vehicle position correction: A vehicular blockchain networks-based gps error sharing framework," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 2, pp. 898–912, 2021.

[4] M. Welsh and G. Mainland, "Programming sensor networks using abstract regions." in *Proc. USENIX Symposium on Networked Systems Design and Implementation*, vol. 4. San Francisco, California, USA, 2004, pp. 3–3.

[5] G. Luo, H. Zhou, N. Cheng, Q. Yuan, J. Li, F. Yang, and X. Shen, "Software-defined cooperative data sharing in edge computing assisted 5g-vanet," *IEEE Trans. on Mobile Computing*, vol. 20, no. 3, pp. 1212–1229, 2019.

[6] E. F. Maleki, L. Mashayekhy, and S. M. Nabavinejad, "Mobility-aware computation offloading in edge computing using machine learning," *IEEE Trans. on Mobile Computing*, vol. 22, no. 1, pp. 328–340, 2023.

[7] W. Chu, H. Zhao, Z. Jin, and Z. Hu, "Adaptive data sharing and computation offloading in cloud-edge computing with resource constraints," in *IEEE Int. Conf. Syst., Man, and Cybernetics*, 2020, pp. 2842–2849.

[8] 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 Journal*, vol. 6, no. 3, pp. 5853–5863, 2019.

[9] M. Siew, D. Cai, L. Li, and T. Q. Quek, "Dynamic pricing for resource-quota sharing in multi-access edge computing," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 4, pp. 2901–2912, 2020.

[10] A. Grange, I. Kacem, and S. Martin, "Algorithms for the bin packing problem with overlapping items," *Computers & Industrial Engineering*, vol. 115, pp. 331–341, 2018.

[11] S. Rampersaud and D. Grosu, "Sharing-aware online algorithms for virtual machine packing in cloud environments," in *Proc. 8th IEEE Int. Conf. on Cloud Computing*, 2015, pp. 718–725.

[12] ——, "Sharing-aware online virtual machine packing in heterogeneous resource clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 7, pp. 2046–2059, 2016.

[13] S. Rabinia, H. Mehryar, M. Brocanelli, and D. Grosu, "Data sharing-aware task allocation in edge computing systems," in *Proc. IEEE Int. Conf. on Edge Computing*, 2021, pp. 60–67.

[14] K. Katsalis, T. G. Papaioannou, N. Nikaein, and L. Tassiulas, "Sla-driven vm scheduling in mobile edge computing," in *2016 IEEE 9th International Conference on Cloud Computing*, 2016, pp. 750–757.

[15] R. Mahmud, K. Ramamohanarao, and R. Buyya, "Latency-aware application module management for fog computing environments," *ACM Trans. Internet Technol.*, vol. 19, no. 1, nov 2018.

[16] Z. Wang, S. Zheng, Q. Ge, and K. Li, "Online offloading scheduling and resource allocation algorithms for vehicular edge computing system," *IEEE Access*, vol. 8, pp. 52 428–52 442, 2020.

[17] S. F. Abedin, M. G. R. Alam, S. M. A. Kazmi, N. H. Tran, D. Niyato, and C. S. Hong, "Resource allocation for ultra-reliable and enhanced mobile broadband iot applications in fog network," *IEEE Transactions on Communications*, vol. 67, no. 1, pp. 489–502, 2019.

[18] P. Erdös and A. Rényi, "On random graphs i," *Publicationes Mathematicae Debrecen*, vol. 6, p. 290, 1959.