

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

Sanaz Rabinia, *Student Member, IEEE*, Daniel Grosu, *Senior Member, IEEE*

Abstract—Edge computing represents an approach in network architecture, allowing data processing closer to the data source or edge of the network. Transferring the data from the offloaded tasks by the end-user devices to edge devices can slow down the network due to increased latency and congestion. Many of these tasks offloaded to edge servers that perform computation to analyze the sensing data, may process the same data items. In order to reduce the traffic in the networks and the number of edge servers, we introduce task allocation algorithms that take into account the sharing of data among the tasks offloaded to the same server. Given that edge computing facilitates real-time processing, in this paper we design heuristic online algorithms for task allocation in edge computing systems. We perform an extensive performance analysis by comparing our proposed algorithms with several sharing-oblivious and sharing-aware 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. Compared to a sharing-aware baseline algorithm, our proposed algorithms are able to reduce the amount of data transferred in the network by 18.1% to 86.9% and the number of utilized servers by 35.4% to 88%. We also augment the proposed data sharing-aware algorithms with a local search phase that iteratively attempts to improve the solutions obtained by our data sharing-aware algorithms. It does this by exploring the neighborhood of the current solution and making small modifications. Our extensive experimental performance analysis shows that the algorithms augmented with local search reduce the number of utilized servers by 9.1% to 66.7% compared to the data sharing-aware online algorithms at the expense of a small increase in the amount of data transferred in the network and a small increase in execution time.

Index Terms—Edge Computing, Task Allocation, Online Algorithms, Data Sharing, Local Search.



1 INTRODUCTION

Edge computing is a distributed computing paradigm that brings computation and data processing closer to the location where it is needed, such as IoT devices, sensors, or end-user devices. In recent years, the adoption of edge computing is growing due to its ability to enable real-time data processing, reduce latency, and minimize bandwidth usage, which is crucial for applications like IoT and autonomous systems. This results in improving response times and enhancing overall system efficiency. One of the significant challenges in edge computing is dealing with the large amount of data transferred in the edge network. Edge computing allows for scalable solutions in deploying services and applications by leveraging data processing on edge devices. One of the main factors in efficient deployment on edge servers is considering the sharing of data among tasks that can be offloaded to the same server.

For example, typical tasksets of autonomous vehicles, include various tasks that use a camera frame as input for lane detection, obstacle detection, segmentation, and localization [1]. The same camera frame can be used by a task for face recognition and by another task for object detection [2]. A single audio recording might be analyzed by one task for voice recognition purposes and by another for detecting background noise levels [3]. 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 edge systems, which generally have limited computational and memory capacities. In several real-world edge-based applications, tasks arrive online and therefore the allocation has to be decided by an online sharing-aware allocation algorithm. Thus, we focus on designing heuristic sharing-aware online task allocation algorithms for such applications. More specifically, we target data-intensive yet moderately latency-sensitive applications, such as cross-camera video analytics requiring online data sharing at the edge [5], federated model updates at the edge, where data-sharing workloads arrive sequentially [6], and multi-user augmented/virtual reality (AR/VR) with online streaming data [7], where incoming tasks frequently reuse shared datasets (e.g., preprocessed image features, intermediate model parameters, or cached sensor data) distributed across multiple edge servers.

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 heuristic online algorithms for solving the ODSA 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

• S. Rabinia and D. Grosu are with the Dept. of Computer Science, Wayne State Univ., Detroit, MI 48202. E-mail: {srabin, dgrosu}@wayne.edu.

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 is an extended version of [8] in which we presented preliminary results on our data sharing-aware online algorithms. Different from our previous work [8], in this paper we design three heuristic online algorithms augmented with a local search phase that iteratively attempts to improve the solutions obtained by our proposed online algorithms [8] by exploring the neighborhood of the current solution and making minor changes.

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 heuristic 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 use it to decide the allocation of tasks to edge servers.
- We augment the three data sharing-aware algorithms (DSA-FF, DSA-BF, and DSA-WF) with a local search phase to obtain three new algorithms (DSA-FF-LS, DSA-BF-LS, and DSA-WF-LS). For the local search phase, we define two main operations (Swap and Move) that explore the neighborhood of the current solution to identify a better solution.
- We perform an extensive performance analysis by comparing our proposed algorithms with several *sharing-oblivious baseline online algorithms*. The baseline algorithms 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 DSA-FF, DSA-BF, and DSA-WF 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. We also compare with a *baseline sharing-aware algorithm*. The results show that DSA-FF, DSA-BF, and DSA-WF are able to reduce the amount of data transferred in the network by 18.1% to 86.9% and the number of utilized servers by 35.4% to 88% compared to the sharing-aware baseline algorithm.
- We also compare the algorithms augmented with local search (DSA-FF-LS, DSA-BF-LS, and DSA-WF-LS) against DSA-FF, DSA-BF, and DSA-WF. The results show that the algorithms augmented with the local search significantly reduce the number of utilized

servers by 9.1% to 66.7%, compared to DSA-FF, DSA-BF, and DSA-WF. Although the addition of local search phase leads to a slight increase in execution time of up to 2.29 times relative to the algorithms without local search, overall, the execution time of the algorithms augmented with local search does not exceed 4 milliseconds. There is also a slight increase of less than 10% in the data transferred in the network, which is the tradeoff of obtaining a significant reduction in the number of utilized servers.

2 RELATED WORK

In the following, we discuss the related work on task allocation in edge computing and task allocation with data sharing techniques in edge computing.

2.1 Task Allocation in Edge Computing

In recent years, task allocation in edge computing has been the focus of extensive research. Yang et al. [9] proposed energy consumption models of different types of equipment in Mobile Edge Computing and studied the placement of cloudlets on the edge nodes. Ullah et al. [10] applied a Double Deep Q-Network (DDQN) algorithm to dynamically make offloading decisions by analyzing task constraints, resource utilization, and the current status of the edge-cloud network. Xiao and Yang [11] proposed a task allocation method to allocate tasks to edge nodes that optimizes the total time and resource utilization. Sahni et al. [12] proposed a data-aware task allocation problem to jointly schedule task and network flows in collaborative edge computing. Their mathematical model minimized the overall completion time of the application. Chen et al. [13] designed a task allocation scheme, which assigns more important tasks to more powerful edge devices to maximize the overall performance. Liu et al. [14] studied the joint task offloading and resource allocation for device-edge-cloud collaboration by formulating an optimization problem with the objective of minimizing the task latency. Zhu et al. [15] designed a workflow scheduler to use in a hybrid edge-cloud network by developing a new graph neural network (GNN)-based representations for task embedding and designing a proximal policy optimization. He et al. [16] introduced a two-layer game framework to improve resource efficiency and system responsiveness by developing algorithms that optimize task offloading and resource allocation strategies for both mobile devices and edge servers, ensuring the convergence to Nash equilibrium in both layers. *None of the above studies consider the fact that some tasks may process the same data, and thus, do not take into account the sharing of data when making allocation decisions.*

2.2 Task Allocation with Data Sharing Techniques in Edge Computing

Many studies, applied unique data sharing techniques into their task allocation problems. Luo et al. [17] 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. Chu et al. [18] 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. [19] 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. Zhao et al. [20] proposed an approximation algorithm to maximize data sharing among continuous interval sampling tasks by jointly optimizing task allocation and scheduling of sampling intervals in low-power wireless sensor networks. In our previous work [21], [22], we proposed *offline* data-sharing aware task allocation greedy algorithms for edge computing systems that maximize 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. *None of these works considered the task allocation with the objective of minimizing the number of utilized servers by taking into account the sharing of data among tasks.*

The problem investigated in this paper is a variant of vector bin-packing [23] 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 [24], [25] 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 [24], [25] 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.

Organization. The rest of the paper is organized as follows. Section 3 formulates the online data sharing-aware task allocation problem. Section 4 describes the proposed online data sharing-aware algorithms. Section 5 introduces the online algorithms augmented with the local search phase. Section 6 presents the experimental results. Section 7 concludes the paper and describes our future work.

3 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 $\mathcal{S} = \{S_1, S_2, \dots, S_M\}$ 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

TABLE 1: Notation

Notation	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 GIPS).
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 GIPS).
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 .
γ	Multiplicative coefficient characterizing the maximum allowable increase in the amount of transferred data.
$q_{\max}$	Maximum allowed number of Move & Swap iterations.

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 $\mathcal{T} = \{T_1, T_2, \dots, T_N\}$ of N tasks, where $\mathcal{T}$ has an associated set of D data items, $\mathcal{D} = \{D_1, D_2, \dots, D_D\}$, that are needed to execute the tasks. We denote the size of data item D_l by d_l (in GB), where $l = 1, 2, \dots, D$. 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 notation used in the paper is given in Table 1.

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 server, 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,

$$\delta_j^k = \sum_{l=1}^D a_{jl} \cdot \llbracket a_{jl} = \Delta_l^k \rrbracket \quad (1)$$

where $\llbracket C \rrbracket$ 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 $\llbracket C \rrbracket$ 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,

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

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,

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

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:

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

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

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 ODSA instance. In the following, we give an example of an ODSA instance consisting of a set of six servers $\mathcal{S} = \{S_k | k = 1, \dots, 6\}$ and a set of six tasks $\mathcal{T} = \{T_j | j = 1, \dots, 6\}$ that need data items from a set $\mathcal{D} = \{D_l | l = 1, \dots, 6\}$. 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 2.

TABLE 2: Example of ODSA 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:

$$\begin{matrix} & D_1 & D_2 & D_3 & D_4 & D_5 & D_6 \\ \begin{matrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \end{matrix} & \begin{bmatrix} 2 & 4 & 0 & 5 & 0 & 1 \\ 2 & 0 & 6 & 5 & 0 & 0 \\ 2 & 4 & 0 & 0 & 0 & 1 \\ 2 & 4 & 6 & 5 & 4 & 0 \\ 2 & 4 & 6 & 5 & 4 & 1 \\ 0 & 0 & 0 & 5 & 4 & 1 \end{bmatrix} \end{matrix} \quad (6)$$

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 .

4 ONLINE ALGORITHMS FOR ODSA

In this section, we design online data sharing-aware task allocation algorithms that solve the ODSA problem. The main objective of the algorithms for ODSA 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.

4.1 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

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:      $\delta_j^k \leftarrow \sum_{l=1}^D a_{jl} \cdot \llbracket a_{jl} = \Delta_l^k \rrbracket$ 
5:     if ( $([\tilde{C}_k^m, \tilde{C}_k^u] - [\sum_{l=1}^D a_{jl} - \delta_j^k, r_j^u]) \geq [0, 0]$ ) then
6:        $\Delta^k \leftarrow \Delta^k \boxplus [A]_{j,*}$ 
7:        $S_k \leftarrow S_k \cup \{T_j\}$ 
8:        $[\tilde{C}_k^m, \tilde{C}_k^u] \leftarrow ([\tilde{C}_k^m, \tilde{C}_k^u] - [\sum_{l=1}^D a_{jl} - \delta_j^k, r_j^u])$ 
9:       break
10:    else
11:       $k \leftarrow k + 1$ 
12: at task  $T_i$  completion on its allocated server  $S_k$  do
13:    $S_k \leftarrow S_k \setminus \{T_i\}$ 
14:    $\Delta^k \leftarrow \Delta^k \boxminus [A]_{i,*}$ 
15:    $\delta_i^k \leftarrow \sum_{l=1}^D a_{il} \cdot \llbracket a_{il} = \Delta_l^k \rrbracket$ 
16:    $[\tilde{C}_k^m, \tilde{C}_k^u] \leftarrow ([\tilde{C}_k^m, \tilde{C}_k^u] + [\sum_{l=1}^D a_{il} - \delta_i^k, r_i^u])$ 

```

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_i on its allocated server S_k , DSA-FF removes T_i from set 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_i^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 ODSA instance described in Section 3. 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 $[\tilde{C}_1^m, \tilde{C}_1^u] - [(a_{21} + a_{23} + a_{24}) - \delta_1^2, r_2^u] = [13, 14] - [(2 + 6 + 5) - (2 + 5), 6] = [7, 8] \geq [0, 0]$, 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, $\delta_1^2 = 2 + 5 = 7$ 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 ODSA 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 (greater than that obtained by DSA-FF) and the number of utilized servers is 4.

Time complexity. The worst case time complexity of DSA-

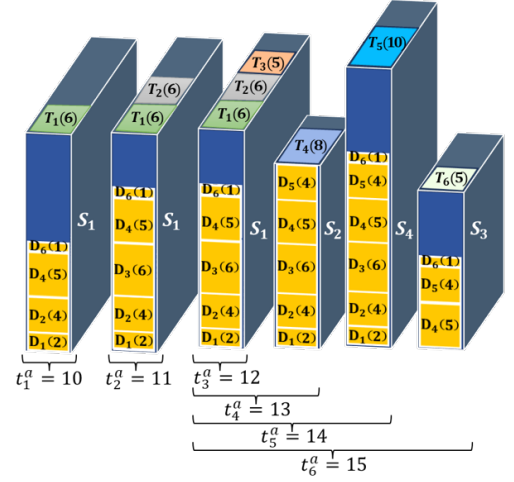


Fig. 1: **DSA-FF execution example** (the vertical dimension corresponds to the memory, while the depth dimension corresponds to the CPU; the numbers in parentheses are the size of CPU requests and the size of data items).

FF at a task arrival is $O(MD)$. 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 $O(D)$, and is due to the update of the data allocation vector.

4.2 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 of server S_k relative to task T_j , as follows,

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

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

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

```

1: at task  $T_j$  arrival do
2:   for  $k = 1, \dots, |\mathcal{S}|$  do
3:      $\delta_j^k \leftarrow \sum_{l=1}^D a_{jl} \cdot \llbracket a_{jl} = \Delta_l^k \rrbracket$ 
4:     if  $([\tilde{C}_k^m, \tilde{C}_k^u] - [\sum_{l=1}^D a_{jl} - \delta_j^k, r_j^u]) \geq [0, 0])$  then
5:        $E_j^k \leftarrow [\alpha(\tilde{C}_k^u - r_j) + \beta(\tilde{C}_k^m - (\sum_{l=1}^D a_{jl} - \delta_j^k))]^{-\frac{1}{2}}$ 
6:     else
7:        $E_j^k \leftarrow 0$ 
8:    $\tilde{k} \leftarrow \operatorname{argmax}_k \{E_j^k\}$ 
9:    $\Delta_{\tilde{k}}^k \leftarrow \Delta_{\tilde{k}}^k \boxplus [A]_{j,*}$ 
10:   $S_{\tilde{k}} \leftarrow S_{\tilde{k}} \cup \{T_j\}$ 
11:   $[\tilde{C}_{\tilde{k}}^m, \tilde{C}_{\tilde{k}}^u] \leftarrow ([\tilde{C}_{\tilde{k}}^m, \tilde{C}_{\tilde{k}}^u] - [\sum_{l=1}^D a_{jl} - \delta_j^k, r_j^u])$ 
12: at task  $T_i$  completion on its allocated server  $S_k$  do
13:   $S_k \leftarrow S_k \setminus \{T_i\}$ 
14:   $\Delta_k^k \leftarrow \Delta_k^k \boxminus [A]_{i,*}$ 
15:   $\delta_i^k \leftarrow \sum_{l=1}^D a_{il} \cdot \llbracket a_{il} = \Delta_l^k \rrbracket$ 
16:   $[\tilde{C}_k^m, \tilde{C}_k^u] \leftarrow ([\tilde{C}_k^m, \tilde{C}_k^u] + [\sum_{l=1}^D a_{il} - \delta_i^k, r_i^u])$ 

```

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.

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 $\tilde{C}_k^m - (\sum_{l=1}^D a_{jl} - \delta_j^k)$, 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

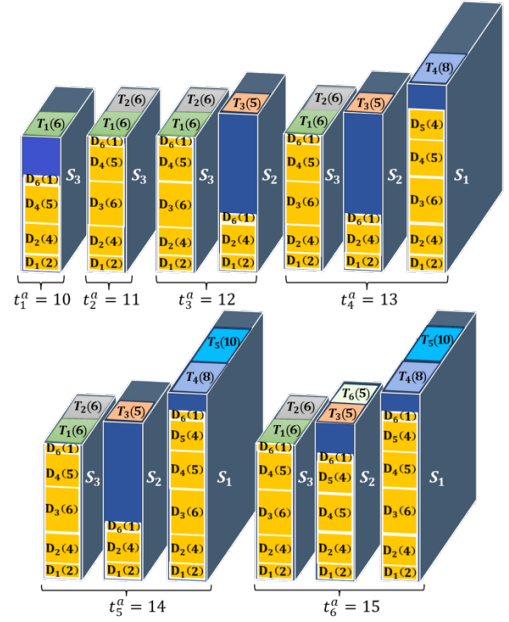


Fig. 2: **DSA-BF execution example** (the vertical dimension corresponds to the memory while the depth dimension corresponds to the CPU; the numbers in parentheses are the size of CPU requests and the size of data items).

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 $\Delta_{\tilde{k}}^k$ 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_i on its allocated server S_k , DSA-BF removes T_i from set 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_i 's allocated resources (Line 16).

DSA-BF: Illustrative example. To illustrate how DSA-BF works, we consider the example of ODSA instance described in Section 3. 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_1^k for all servers and since all servers have enough capacity it calculates their efficiency metric (with $\alpha = \beta = 0.5$) (Line 5). Thus, $E_1^1 = [0.5(12 - 6) + 0.5(18 - 12)]^{-1/2} = 0.408$ 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 $[\tilde{C}_3^m, \tilde{C}_3^u] - [(a_{11} + a_{12} + a_{14} + a_{16}) - \delta_1^1, r_1^u] = [18, 12] - [(2 + 4 + 5 + 1) - 0, 6] = [6, 6] \geq [0, 0]$. 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 ODSA 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. Thus, DSA-BF transfers less data and utilizes fewer servers.

Time complexity. The worst case time complexity of DSA-BF at a task arrival is $O(MD)$. 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 $O(D)$, and is due to the update of the data allocation vector.

4.3 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

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

```

1: at task  $T_j$  arrival do
2:   for  $k = 1, \dots, |S|$  do
3:      $\delta_j^k \leftarrow \sum_{l=1}^D a_{jl} \cdot \llbracket a_{jl} = \Delta_l^k \rrbracket$ 
4:     if  $([\tilde{C}_k^m, \tilde{C}_k^u] - [\sum_{l=1}^D a_{jl} - \delta_j^k, r_j^u] \geq [0, 0])$  then
5:        $E_j^k \leftarrow [\alpha(\tilde{C}_k^u - r_j) + \beta(\tilde{C}_k^m - (\sum_{l=1}^D a_{jl} - \delta_j^k))]^{-\frac{1}{2}}$ 
6:     else
7:        $E_j^k \leftarrow 0$ 
8:    $\tilde{k} \leftarrow \operatorname{argmin}_k \{E_j^k\}$ 
9:    $\Delta_{\tilde{k}}^k \leftarrow \Delta_{\tilde{k}}^k \sqcup [A]_{j,*}$ 
10:   $S_{\tilde{k}} \leftarrow S_{\tilde{k}} \cup \{T_j\}$ 
11:   $[\tilde{C}_{\tilde{k}}^m, \tilde{C}_{\tilde{k}}^u] \leftarrow ([\tilde{C}_{\tilde{k}}^m, \tilde{C}_{\tilde{k}}^u] - [\sum_{l=1}^D a_{jl} - \delta_j^{\tilde{k}}, r_j^u])$ 
12: at task  $T_i$  completion on its allocated server  $S_k$  do
13:   $S_k \leftarrow S_k \setminus \{T_i\}$ 
14:   $\Delta_k^k \leftarrow \Delta_k^k \sqcup [A]_{i,*}$ 
15:   $\delta_i^k \leftarrow \sum_{l=1}^D a_{il} \cdot \llbracket a_{il} = \Delta_l^k \rrbracket$ 
16:   $[\tilde{C}_k^m, \tilde{C}_k^u] \leftarrow ([\tilde{C}_k^m, \tilde{C}_k^u] + [\sum_{l=1}^D a_{il} - \delta_i^k, r_i^u])$ 

```

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). Since DSA-WF is similar to DSA-BF, due to space limitations we do not present an illustrative example of DSA-WF's execution.

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

5 DSA-X-LS ALGORITHMS: DSA-X WITH LOCAL SEARCH

In this section, we focus on improving the solutions obtained by the DSA-X algorithms, where X is FF, BF, or WF. This is achieved by augmenting these algorithms with a local search phase. This approach seeks to further reduce the total number of utilized servers at the expense of a small increase in the the amount of data transferred in the network. Since the local search phase is the same for all DSA-X algorithms we present the new algorithms as a single algorithm called DSA-X-LS.

The DSA-X-LS consists of two phases: (1) Execute DSA-X, and (2) Local search. In *Phase 1*, it executes DSA-X to obtain an initial solution, and then, in *Phase 2*, it iteratively attempts to improve the solution by exploring the neighborhood of the current solution by making small modifications to it. It evaluates each neighboring solution to determine the improvement of the solution. *Phase 2* employs two fundamental operations: Move and Swap, which are

Algorithm 4 Data Sharing-Aware-Local Search (DSA-X-LS)

```

1: at task  $T_j$  arrival do
2:   Phase 1: Execute DSA-X
3:   if X=FF execute Lines 2-11 of DSA-FF
4:   if X=BF execute Lines 2-11 of DSA-BF
5:   if X=WF execute Lines 2-11 of DSA-WF
6:   Phase 2: Local Search
7:    $q \leftarrow 0$ 
8:   for all  $k \in S$  do
9:     for all  $i \in S_k$  do
10:      for all  $k' \in S$  do
11:        if  $k \neq k'$  then
12:           $q \leftarrow q + 1$ 
13:          Move( $S_k, T_i, S_{k'}$ )
14:          if  $q = q_{\max}$  then
15:            break
16:          for all  $j \in S_{k'}$  do
17:             $q \leftarrow q + 1$ 
18:            Swap( $S_k, T_i, S_{k'}, T_j$ )
19:            if  $q = q_{\max}$  then
20:              break
21: at task  $T_i$  completion on its allocated server  $S_k$  do
22:   $S_k \leftarrow S_k \setminus \{T_i\}$ 
23:   $\Delta_k^k \leftarrow \Delta_k^k \sqcup [A]_{i,*}$ 
24:   $\delta_i^k \leftarrow \sum_{l=1}^D a_{il} \cdot \llbracket a_{il} = \Delta_l^k \rrbracket$ 
25:   $[\tilde{C}_k^m, \tilde{C}_k^u] \leftarrow ([\tilde{C}_k^m, \tilde{C}_k^u] + [\sum_{l=1}^D a_{il} - \delta_i^k, r_i^u])$ 

```

given in Algorithms 5 and 6, respectively. To explore nearby solutions to the current one, **Move** performs a single move of a task from its currently allocated server to another server, while **Swap** swaps two tasks from two different servers. If the current solution is improved, the improved solution replaces the current solution, otherwise it proceeds to the next iteration. This is continued until the number of performed **Move** and **Swap** operations reaches a set value.

DSA-X-LS, given in Algorithm 4, is executed when a new task arrives (Line 1). In *Phase 1*, it executes one of the DSA-X algorithms to obtain an initial solution (Lines 2-4). In *Phase 2*, it performs a local search procedure starting with the solution obtained in Phase 1. It first initializes a counter q of local search operations (Line 5) and then it starts the local search operations (Lines 6-18). For each source server S_k , and destination server $S_{k'}$, it first increments the operation counter q and applies the **Move** operation on task T_i , which moves T_i from the source to the server $S_{k'}$ if this improves the solution (Lines 6-11). After this, it increments operation counter q and applies the **Swap** operation which swaps tasks T_i from server S_k with task T_j from server $S_{k'}$ if the swap improves the solution (Lines 15-16). After each **Move** and **Swap** operations the algorithm checks if the value of the operation counter q reached the given maximum value $q_{\max}$, where $q_{\max}$ is a system-defined parameter that specifies how many **Move** and **Swap** operations the algorithm is allowed to execute (e.g., $q_{\max}$ is set to 10 in our experiments). If it does, the local search phase is terminated and the algorithm waits for the arrival of another task (Lines 12-13 and 17-18).

After the completion of task T_i on its allocated server S_k , the algorithm deallocates task T_i from its allocated server, updates the data allocation vector Δ^k , updates the data sharing value δ_i^k , and finally frees up server S_k 's resources (Lines 19-23).

Move Operation. The **Move** operation (Algorithm 5) attempts to relocate a single task from one server to another. It is particularly useful when tasks in one server create inefficiencies due to poor packing. For example, if a task is the only task on a server, and there is still space available for the task on other already allocated servers, it can be moved to that server, thereby reducing the number of utilized servers. These particular types of situation occur because of task completions. The input of the **Move** algorithm is a triple including a source server S_k , a destination server $S_{k'}$, and the task T_i to be moved from server S_k .

First, **Move** determines the current number of utilized servers, e , and the current total amount of data stored across all servers, m , and the current total amount of data transferred among servers, z (Lines 1-3). Then, it checks if the destination server $S_{k'}$ has enough CPU and memory capacity to host task T_i (Line 4). If $S_{k'}$ has enough capacity, it deallocates task T_i from its current server S_k , restores the CPU and memory capacity for server S_k after deallocating task T_i , allocates task T_i to server $S_{k'}$, and updates the CPU and memory capacity of server $S_{k'}$ (Lines 5-12). Next, it checks if the current number of utilized servers and the current total amount of data across all servers after moving task T_i are less than the initial number of utilized servers and total amount of data, respectively. It also checks if the total amount of data transferred among servers is less than

Algorithm 5 $\text{Move}(S_k, T_i, S_{k'})$

Input: source server: S_k ;
destination server: $S_{k'}$;
task: $T_i \in S_k$.

- 1: $e \leftarrow$ current number of utilized servers
- 2: $m \leftarrow$ current total amount of data stored across all servers
- 3: $z \leftarrow$ current total amount of data transferred among servers
- 4: **if** $[\tilde{C}_{k'}^m, \tilde{C}_{k'}^u] - [\sum_{l=1}^D a_{il} - \delta_i^k, r_i^u] \geq [0, 0]$ **then**
- 5: $S_k \leftarrow S_k \setminus \{T_i\}$
- 6: $\Delta^k \leftarrow \Delta^k \ominus [A]_{i,*}$
- 7: $\delta_i^k \leftarrow \sum_{l=1}^D a_{il} \cdot \llbracket a_{il} = \Delta_l^k \rrbracket$
- 8: $[\tilde{C}_k^m, \tilde{C}_k^u] \leftarrow [\tilde{C}_k^m, \tilde{C}_k^u] + [\sum_{l=1}^D a_{il} - \delta_i^k, r_i^u]$
- 9: $\delta_i^k \leftarrow \sum_{l=1}^D a_{il} \cdot \llbracket a_{il} = \Delta_l^k \rrbracket$
- 10: $\Delta^{k'} \leftarrow \Delta^{k'} \oplus [A]_{i,*}$
- 11: $S_{k'} \leftarrow S_{k'} \cup \{T_i\}$
- 12: $[\tilde{C}_{k'}^m, \tilde{C}_{k'}^u] \leftarrow [\tilde{C}_{k'}^m, \tilde{C}_{k'}^u] - [\sum_{l=1}^D a_{il} - \delta_i^{k'}, r_i^{k'}]$
- 13: **if** current number of utilized servers $< e$ **and** current total amount of data stored across all servers $< m$ **and** $z + [\sum_{l=1}^D a_{il} - \delta_i^{k'}] \leq z \cdot \gamma$ **then**
- 14: $e \leftarrow e - 1$
- 15: $m \leftarrow m + [\sum_{l=1}^D a_{il} - \delta_i^k] - [\sum_{l=1}^D a_{il} - \delta_i^{k'}]$
- 16: $z \leftarrow z + [\sum_{l=1}^D a_{il} - \delta_i^{k'}]$
- 17: **else**
- 18: $S_{k'} \leftarrow S_{k'} \setminus \{T_i\}$
- 19: $\Delta^{k'} \leftarrow \Delta^{k'} \ominus [A]_{i,*}$
- 20: $\delta_i^{k'} \leftarrow \sum_{l=1}^D a_{il} \cdot \llbracket a_{il} = \Delta_l^{k'} \rrbracket$
- 21: $[\tilde{C}_{k'}^m, \tilde{C}_{k'}^u] \leftarrow [\tilde{C}_{k'}^m, \tilde{C}_{k'}^u] + [\sum_{l=1}^D a_{il} - \delta_i^{k'}, r_i^{k'}]$
- 22: $\delta_i^{k'} \leftarrow \sum_{l=1}^D a_{il} \cdot \llbracket a_{il} = \Delta_l^{k'} \rrbracket$
- 23: $\Delta^k \leftarrow \Delta^k \oplus [A]_{i,*}$
- 24: $S_k \leftarrow S_k \cup \{T_i\}$
- 25: $[\tilde{C}_k^m, \tilde{C}_k^u] \leftarrow [\tilde{C}_k^m, \tilde{C}_k^u] - [\sum_{l=1}^D a_{il} - \delta_i^k, r_i^u]$

a threshold $z \cdot \gamma$ (Line 13). γ is a parameter set by the system administrator which specifies the allowable increase in data transfer across edge servers. If this is the case, then it reduces the number of utilized servers e by one, updates the total amount of memory used, and total amount of data transferred among servers (Lines 14-16). Otherwise, it deallocates T_i from server $S_{k'}$, updates the CPU and memory capacity of server $S_{k'}$, allocates T_i back to S_k , and updates the CPU and memory capacity of server S_k (Lines 18-25).

Swap Operation. The **Swap** operation (Algorithm 6) performs task exchanges between two servers, which can lead to better resource utilization. Swapping allows for more substantial changes in the solution, as it considers the interaction between two tasks and how that affects the overall solution. It is meant for exploration of larger changes since it can lead to more significant improvements by changing the relationship between two tasks and their servers. Note that the **Swap** operation does not change the number of utilized servers. The input of this algorithm is a quadruple including a source server S_k , destination server $S_{k'}$, along with tasks T_i from S_k and T_j from $S_{k'}$.

First, **Swap** determines the current total amount of data stored across all servers, m , and across all servers excluding the source and destination servers, m' (Line 1-2). It also determines the current total amount of data transferred among servers, z (Line 3). It calculates m_1 , the amount of data on server S_k after deallocating T_i and allocating T_j , and m_2 the amount of data on server $S_{k'}$ after deallocating T_j and

Algorithm 6 Swap($S_k, T_i, S_{k'}, T_j$)

Input: source server: S_k ;
destination server: $S_{k'}$;
task: $T_i \in S_k$;
task: $T_j \in S_{k'}$.

- 1: $m \leftarrow$ current total amount of data stored across all servers
- 2: $m' \leftarrow$ current total amount of data stored across all servers except S_k and $S_{k'}$
- 3: $z \leftarrow$ current total amount of data transferred among servers
- 4: $m_1 = [\sum_{l=1}^D a_{il} - \delta_i^k] - [\sum_{l=1}^D a_{jl} - \delta_j^k]$
- 5: $m_2 = [\sum_{l=1}^D a_{jl} - \delta_j^{k'}] - [\sum_{l=1}^D a_{il} - \delta_i^{k'}]$
- 6: **if** $[\tilde{C}_k^m, \tilde{C}_k^u] + [m_1, r_i^u - r_j^u] \geq [0, 0]$ **and** $[\tilde{C}_{k'}^m, \tilde{C}_{k'}^u] + [m_2, r_j^u - r_i^u] \geq [0, 0]$ **then**
- 7: $S_k \leftarrow S_k \setminus \{T_i\}$
- 8: $\Delta_i^k \leftarrow \Delta_i^k \boxminus [A]_{i,*}$
- 9: $\delta_i^k \leftarrow \sum_{l=1}^D a_{il} \cdot \llbracket a_{il} = \Delta_l^k \rrbracket$
- 10: $[\tilde{C}_k^m, \tilde{C}_k^u] \leftarrow [\tilde{C}_k^m, \tilde{C}_k^u] + [\sum_{l=1}^D a_{il} - \delta_i^k, r_i^u]$
- 11: $\delta_j^{k'} \leftarrow \sum_{l=1}^D a_{jl} \cdot \llbracket a_{jl} = \Delta_l^{k'} \rrbracket$
- 12: $\Delta_j^{k'} \leftarrow \Delta_j^{k'} \boxplus [A]_{j,*}$
- 13: $S_{k'} \leftarrow S_{k'} \cup \{T_j\}$
- 14: $[\tilde{C}_{k'}^m, \tilde{C}_{k'}^u] \leftarrow [\tilde{C}_{k'}^m, \tilde{C}_{k'}^u] - [\sum_{l=1}^D a_{jl} - \delta_j^{k'}, r_j^u]$
- 15: $S_k \leftarrow S_k \cup \{T_j\}$
- 16: $\Delta_i^{k'} \leftarrow \Delta_i^{k'} \boxminus [A]_{i,*}$
- 17: $\delta_j^{k'} \leftarrow \sum_{l=1}^D a_{jl} \cdot \llbracket a_{jl} = \Delta_l^{k'} \rrbracket$
- 18: $[\tilde{C}_{k'}^m, \tilde{C}_{k'}^u] \leftarrow [\tilde{C}_{k'}^m, \tilde{C}_{k'}^u] + [\sum_{l=1}^D a_{jl} - \delta_j^{k'}, r_j^u]$
- 19: $\delta_i^{k'} \leftarrow \sum_{l=1}^D a_{il} \cdot \llbracket a_{il} = \Delta_l^{k'} \rrbracket$
- 20: $\Delta_i^{k'} \leftarrow \Delta_i^{k'} \boxplus [A]_{i,*}$
- 21: $S_{k'} \leftarrow S_{k'} \setminus \{T_j\}$
- 22: $[\tilde{C}_{k'}^m, \tilde{C}_{k'}^u] \leftarrow [\tilde{C}_{k'}^m, \tilde{C}_{k'}^u] - [\sum_{l=1}^D a_{il} - \delta_i^{k'}, r_i^u]$
- 23: **if** $m' + (m_1 + m_2) < m$ **and** $z + [\sum_{l=1}^D a_{jl} - \delta_j^k] + [\sum_{l=1}^D a_{il} - \delta_i^{k'}] \leq z \cdot \gamma$ **then**
- 24: $m \leftarrow m' + (m_1 + m_2)$
- 25: $z \leftarrow z + [\sum_{l=1}^D a_{jl} - \delta_j^k] + [\sum_{l=1}^D a_{il} - \delta_i^{k'}]$
- 26: **else**
- 27: $\delta_i^k \leftarrow \sum_{l=1}^D a_{il} \cdot \llbracket a_{il} = \Delta_l^k \rrbracket$
- 28: $\Delta_i^k \leftarrow \Delta_i^k \boxplus [A]_{i,*}$
- 29: $S_k \leftarrow S_k \cup \{T_i\}$
- 30: $[\tilde{C}_k^m, \tilde{C}_k^u] \leftarrow [\tilde{C}_k^m, \tilde{C}_k^u] - [\sum_{l=1}^D a_{il} - \delta_i^k, r_i^u]$
- 31: $S_{k'} \leftarrow S_{k'} \setminus \{T_j\}$
- 32: $\Delta_j^{k'} \leftarrow \Delta_j^{k'} \boxminus [A]_{j,*}$
- 33: $\delta_j^{k'} \leftarrow \sum_{l=1}^D a_{jl} \cdot \llbracket a_{jl} = \Delta_l^{k'} \rrbracket$
- 34: $[\tilde{C}_{k'}^m, \tilde{C}_{k'}^u] \leftarrow [\tilde{C}_{k'}^m, \tilde{C}_{k'}^u] + [\sum_{l=1}^D a_{jl} - \delta_j^{k'}, r_j^u]$
- 35: $\Delta_i^{k'} \leftarrow \Delta_i^{k'} \boxplus [A]_{i,*}$
- 36: $\delta_j^{k'} \leftarrow \sum_{l=1}^D a_{jl} \cdot \llbracket a_{jl} = \Delta_l^{k'} \rrbracket$
- 37: $S_{k'} \leftarrow S_{k'} \cup \{T_j\}$
- 38: $[\tilde{C}_{k'}^m, \tilde{C}_{k'}^u] \leftarrow [\tilde{C}_{k'}^m, \tilde{C}_{k'}^u] - [\sum_{l=1}^D a_{jl} - \delta_j^{k'}, r_j^u]$
- 39: $S_k \leftarrow S_k \setminus \{T_i\}$
- 40: $\Delta_i^k \leftarrow \Delta_i^k \boxminus [A]_{i,*}$
- 41: $\delta_i^k \leftarrow \sum_{l=1}^D a_{il} \cdot \llbracket a_{il} = \Delta_l^k \rrbracket$
- 42: $[\tilde{C}_k^m, \tilde{C}_k^u] \leftarrow [\tilde{C}_k^m, \tilde{C}_k^u] + [\sum_{l=1}^D a_{il} - \delta_i^k, r_i^u]$

allocating T_i (Lines 4-5). Next, it checks if both the source and destination servers have enough resources to allocate T_j to S_k and allocate T_i to $S_{k'}$, respectively (Line 6). If that is the case, it removes T_i from S_k , restores S_k 's CPU and memory capacity after task T_i deallocation, and adds T_j to server S_k and updates server S_k 's capacity (Lines 7-14). It does the same process for servers $S_{k'}$ and S_k and tasks T_j and T_i , respectively (Lines 15-22). Then, it checks if the sum of current total amount of data stored across all servers and the total amount of data on $S_{k'}$ and S_k (i.e., current total amount of data stored across all servers) is less than the

initial total amount of data stored across all servers. It also checks if the current total amount of data transferred is less than a threshold $z \cdot \gamma$ (Line 23). If that is the case, it updates the current total amount of data stored across all servers and the current total amount of data transferred among servers (Lines 24-25). Otherwise, it adds T_i to $S_{k'}$, updates $S_{k'}$'s CPU and memory capacity after task T_i allocation, and removes T_j from server S_k and restores server S_k 's capacity (Lines 27-34). It does the same thing for the server $S_{k'}$ and tasks T_j and T_i , respectively (Lines 35-42).

Phase 2 (Local Search) integrates Move and Swap operations to iteratively refine the solution. If a Move or Swap leads to a valid and improved solution (fewer number of utilized servers and smaller total amount of data stored in the edge servers), it is accepted and performed. The process continues until no further improvements are found or the maximum number of local search operations $q_{\max}$ is reached. By combining the flexibility of Swap and Move, the proposed framework ensures that the solution converges to a near-optimal one, minimizing both the total number of servers and the overall memory usage. The described algorithms collectively implement a robust local search framework for solving the ODSA problem. The framework systematically explores the solution space, avoiding cycles and ensuring convergence toward near-optimal solutions.

Time complexity. The worst case time complexity DSA-X-LS is given by the worst case time complexity of its two phases. The worst case complexity of *Phase 1* is given by the complexity of DSA-X, which is $O(MD)$ (at a task arrival) and $O(D)$ at the completion of a task. The worst case time complexity of *Phase 2* is determined by the time complexity of both Move and Swap operations. Both, Move and Swap take $O(D)$. Those operations are executed $O(q_{\max})$ times. Thus, the worst case complexity of *Phase 2* is $O(q_{\max}D)$.

6 EXPERIMENTAL ANALYSIS

In this section, we investigate the performance of the proposed online data sharing-aware algorithms, and compare them with baseline online sharing-oblivious 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 [26], [27], [28], [29], and thus, we consider them as baselines in our performance analysis. We also compare the proposed algorithms with a baseline sharing-aware algorithm based on Random-Fit. We implement the algorithms in Java and run the simulation experiments on a system (part of Wayne State University's HPC Grid) equipped with AMD processors, 48 cores at 3.2 GHz, 254 GB of RAM, and 83 GB of local storage. First, we describe the experimental setup and then analyze the results.

6.1 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

TABLE 3: Distributions and parameters used for evaluation ($\mathcal{N}$, $\mathcal{U}$, and Exp denote normal, uniform, and exponential 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
Task's Arrival Time	Exp($\lambda = 1/10$), range [5, 50] seconds
Task's Duration	Exp($\lambda = 1/15$), range [5, 50] seconds
Data Items	100
Data Size	$\mathcal{N}(\mu = 25, \sigma = 4)$, range [1, 50] GB

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 3 provides a summary of the simulation parameters and their distributions used in our experiments.

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:

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

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 and exponential 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 Erdős-Rényi random graph model [30] 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 Erdős-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 Erdős-Rényi model using the *igraph* R package. We generate instances for three different cases:

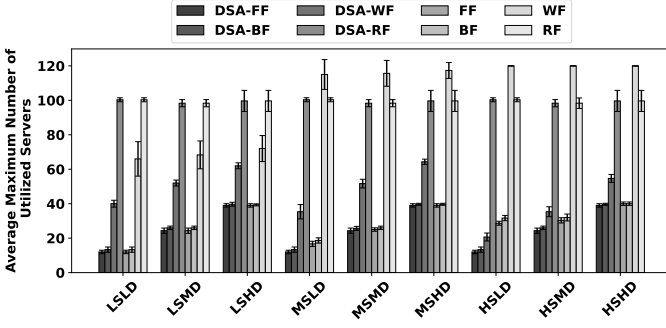
- (i) *Low Sharing* (LS) ($0 \leq p \leq 0.3$), where only a few tasks share data items;
- (ii) *Medium Sharing* (MS) ($0.3 < p \leq 0.6$); and
- (iii) *High Sharing* (HS) ($0.6 < p \leq 1$), where a large number of tasks share data items.

For our experimental results we consider task-matrix $A_{500 \times 100}$, representing a bipartite graph with 500 tasks and 100 data items. The selection of the number of tasks and the number of data items is based on several real-world settings where application-specific tasks share the same data. Those scenarios include typical task sets present in autonomous vehicles [1] and mobile augmented reality applications [31] where tens of tasks of a user analyze the same camera frames. Thus, using a similar task-to-data ratio as in those real-world settings allows us to investigate current and future realistic scenarios with a large number of users, tasks, and data items.

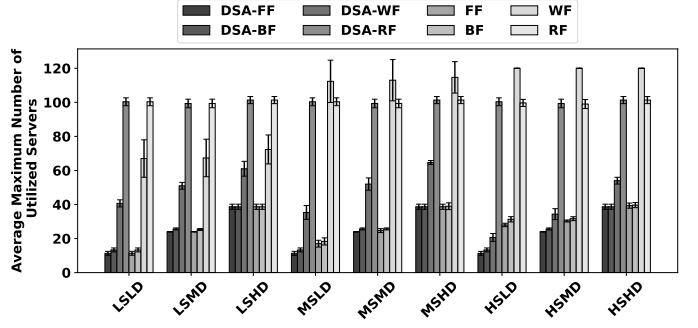
In the experiments, the two parameters of the proposed algorithms specifying the maximum allowable increase in the amount of data transferred in the network for each local search operation, γ , and the maximum allowed number of local search operations, q_{max} , are set to 1.01 and 10, respectively. This means that we allow a maximum of 1% increase in the amount of data transferred in the network for each of local search operation, and a maximum of 10 local search operations per each task allocation.

6.2 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., $\{\text{LS, MS, HS}\} \times \{\text{LD, MD, HD}\}$. Each task is characterized by a certain size of its computational request and the data items needed, and each server has certain CPU and memory capacities, as described in the previous section. We compare the performance of the proposed online data sharing-aware algorithms, DSA-FF, DSA-BF, and DSA-WF, and those augmented with local search, DSA-FF-LS, DSA-BF-LS, and DSA-WF-LS 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. Additionally, we introduce a sharing-aware baseline, Data Sharing-Aware Random Fit (DSA-RF), which selects a server randomly and places the arrived task on that server if the server has sufficient memory and CPU capacity to host the task. Otherwise, if an inactive server is available, a new server is activated and the task is assigned there. If no inactive servers remain and the selected server cannot host the task, then no placement occurs. The algorithm is sharing-aware in the sense that when allocating memory to a task on a server, it checks what data items needed by the task are already in the memory of the server and updates accordingly the remaining memory capacity on that server. We also compare the performance of DSA-RF with its corresponding sharing-oblivious baseline algorithm RF. It is important to mention that these are the only online baseline algorithms we were able to use for comparison. None of the previous works (including ours [22]), whether online, offline, or heuristic approaches share the same objective or



(a) Uniformly distributed task arrival times and task durations.



(b) Exponentially distributed task arrival times and task durations.

Fig. 3: **Sharing-aware vs. baseline sharing-oblivious algorithms: Average maximum number of utilized servers** (DSA-BF, DSA-WF, BF, and WF with $\alpha = \beta = 0.5$).

incorporate data sharing in a comparable manner to this paper. Our previous work [22] considers an offline problem having a different objective, focusing on maximizing the profit obtained from executing tasks while minimizing network traffic in the edge system. Therefore, the algorithms in those works are not directly comparable to the algorithms presented in this paper.

The results were obtained by executing the algorithms three times and computing the average values of the metrics described above and the standard deviation.

Maximum number of utilized servers. For each of the algorithms we record the number of utilized servers at each task arrival and termination and determine the maximum among those values.

Figure 3a shows the average maximum number of servers utilized by the proposed algorithms without the local search phase, their baseline sharing-oblivious counterparts, and the baseline sharing-aware DSA-RF. The results are shown for all nine types of instances with different levels of sharing and demand, and task arrivals and task durations drawn from uniform distribution. For the data sharing-aware algorithms, the average maximum number of utilized servers 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 (12.0 servers on average), while in the case of HSLD the DSA-FF utilizes 12.0 servers on average compared to FF which utilizes 28.67 (i.e., DSA-FF uses 16.67 fewer servers). 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 is obtained by DSA-WF in the case of HSLD, where DSA-WF uses 20.67 servers on average compared to WF which uses 120.0 servers on average (a reduction of 99.33 servers). 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 maximum number of utilized servers 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 utilized servers compared to scenarios where CPU demand is lower.

The proposed sharing-aware algorithms performed significantly better than the sharing-aware baseline DSA-RF across all nine scenarios. Although DSA-RF takes the sharing of data into account when determining the capacities of the servers, it does not actively exploit this information when making placement decisions. Instead, DSA-RF relies purely on random selection to co-locate related tasks, which results in inefficient use of shared memory and a higher number of utilized servers. In contrast, DSA-WF, the worst performer among the proposed algorithms, explicitly incorporates data sharing into its efficiency metric, favoring servers where relevant data already resides. For example, in high-sharing scenarios such as HSLD and HSHD, DSA-WF consistently utilizes fewer servers than DSA-RF baseline, demonstrating that consideration of data sharing leads to better consolidation.

For the sharing-oblivious algorithms, a similar but more pronounced trend is observed. In medium and high-sharing scenarios, WF consistently performs worse than RF. Since WF always selects the server with the maximum available capacity, it tends to place each incoming task on a largely empty server, effectively preventing task co-location and leading to a larger number of active servers. In contrast, RF does not bias placements away from partially loaded servers, allowing co-location to occur by chance. As a result, RF utilizes fewer servers than WF when data sharing is present, despite being unaware of shared data. This contrast further emphasizes the importance of explicitly accounting for data sharing, as even a sharing-aware Worst-Fit strategy (DSA-WF) substantially outperforms the sharing-oblivious algorithms and the sharing-aware baseline (DSA-RF) in scenarios with medium to high data sharing.

Next, we investigate the algorithms' performance sensitivity to task arrival time and task duration distributions. Figure 3b shows the maximum number of utilized servers when the task arrival time and task duration are exponen-

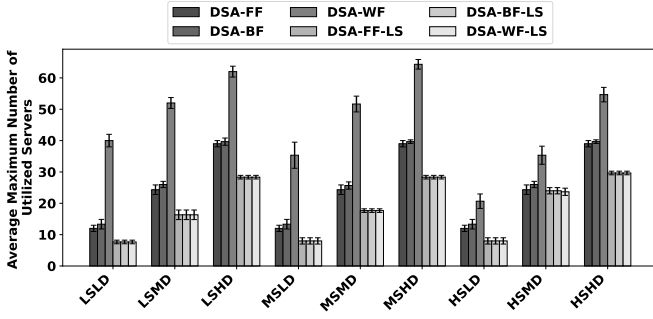


Fig. 4: DSA-X-LS vs. sharing-aware algorithms: Average maximum number of utilized servers. Uniformly distributed task arrival times and task durations. (DSA-BF-LS DSA-WF-LS, DSA-BF, and DSA-WF with $\alpha = \beta = 0.5$).

tially distributed. Compared to the maximum number of utilized servers obtained by the algorithms in the case of uniformly distributed task arrival times and task durations, it changes very little, generally by up to 5% fewer utilized servers in low/medium demand scenarios and by up to 8% fewer utilized servers in high-demand scenarios. Overall, the uniform vs. exponential distribution comparison exhibits only minor differences in the maximum number of utilized servers showing that the proposed algorithms are robust to changes in the distribution of task arrival times and task durations.

Figure 4 shows the average of maximum number of utilized servers by the proposed data sharing-aware algorithms and the algorithms augmented with local search. We observe that DSA-FF obtains the best performance among the algorithms without the local search for all sharing and demand scenarios, followed very closely by DSA-BF. For the data sharing-aware algorithms with local search, we observe a huge reduction in all nine types of instances in terms of maximum number of utilized servers. In the case of LSLD, DSA-WF-LS reduces the average number of utilized servers by more than 75% relative to DSA-WF, and in the worst case (occurring for HSHD), it achieves a reduction of approximately 33.3%. In almost all nine cases, the average number of utilized servers for DSA-BF-LS, DSA-FF-LS, and DSA-WF-LS are roughly the same. This is due to the fact that after applying Swap and Move operations, the local search algorithm tends to consolidate tasks on fewer servers. Thus, there is a consistent trend of task consolidation across all data sharing-aware online algorithms augmented with local search. This also indicates that the algorithms augmented with local search obtain a significant reduction in the number of utilized servers. In the next subsection, we show that this reduction is obtained at the cost of a small increase in the data transferred in the network.

Figure 5 shows the effect of the maximum allowable increase in the data transfers, γ , and the maximum number of move/swap operation, q_{max} , on the maximum number of utilized servers by the proposed algorithms in the HSHD case. The online data sharing-aware algorithms without the local search phase, (DSA-FF, DSA-BF, and DSA-WF), require approximately 39, 39.67, and 54.67 servers on average, respectively. The performance of the algorithms augmented with local search (denoted 'LS' in the figure) varies de-

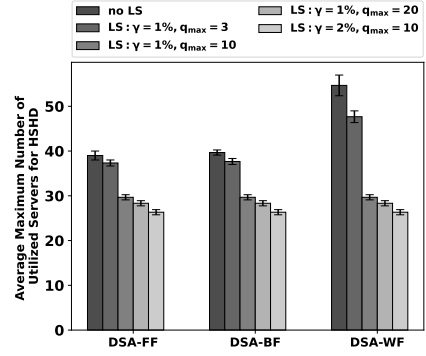


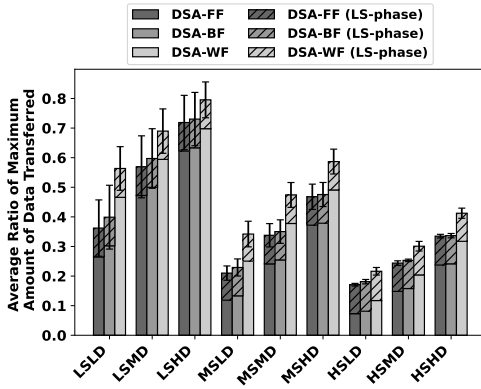
Fig. 5: The effect of γ and q_{max} on the maximum number of utilized servers by the proposed algorithms without local search ('no LS') and those augmented with local search ('LS') (HSHD case and uniform distribution of task arrivals; DSA-BF-LS DSA-WF-LS, DSA-BF, and DSA-WF with $\alpha = \beta = 0.5$).

pending on the choice of γ and q_{max} . With a tighter search ($\gamma = 1\%$, $q_{max} = 3$), the local search phase is unable to sufficiently explore beneficial task reassignments, resulting in a performance close to that of the corresponding baseline algorithm. The largest reduction is obtained for DSA-WF-LS. This is due to the nature of DSA-WF (finds servers with minimum load) it has a proclivity to spread tasks among servers, and thus, even a small number of move/swap operations can result in a huge reduction in the number of servers. In contrast, with a moderate local search depth ($\gamma = 1\%$, $q_{max} = 10$), the algorithm identifies meaningful rearrangements that more effectively co-locate shared data, reducing the number of utilized servers to 29.67 for all three variants, representing an improvement of approximately 24% to 46% relative to their corresponding baselines. Further increasing the search depth ($\gamma = 1\%$, $q_{max} = 20$) yields only marginal additional improvement to around 28.33 servers, indicating diminishing returns once the key beneficial swaps have been discovered. Finally, relaxing the maximum allowable increase in the data transfers ($\gamma = 2\%$, $q_{max} = 10$) provides additional flexibility to execute enabling move operations, achieving the best consolidation with 26.33 utilized servers. This corresponds to reductions of approximately 33% relative to DSA-FF, 34% relative to DSA-BF, and 52% relative to DSA-WF. Overall, these results highlight that increasing γ has a stronger impact on solution quality than solely increasing q_{max} . Allowing a slightly higher amount of data transferred (γ) enables beneficial intermediate moves, which improves consolidation and reduces the number of active servers at the expense of higher volume of traffic in the network. To balance the increase in network traffic and the execution time, we recommend using $\gamma \approx 1\%$, and $q_{max} = 10$.

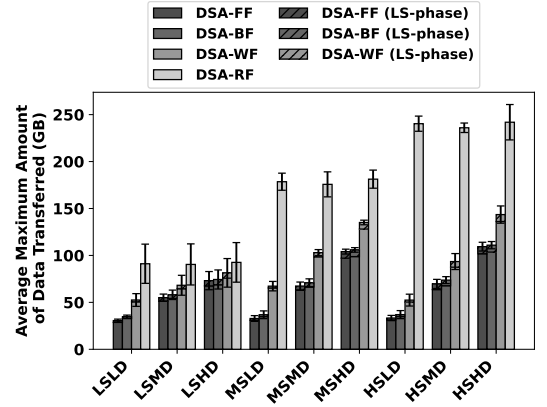
Maximum amount of data transferred in the edge network.

For each of the online algorithms we record the amount of data transferred in the edge network at each task arrival 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 6a shows an overlay bar plot and the ratio for all nine types of instances with different levels of sharing and



(a) Maximum amount of data transferred relative to baseline sharing-oblivious algorithms.



(b) Average maximum amount of data transferred.

Fig. 6: **Relative and average maximum amount of data transferred in the network:** all six proposed algorithms and nine sharing and demand scenarios (DSA-BF-LS, DSA-WF-LS, DSA-BF, DSA-WF, BF, and WF with $\alpha = \beta = 0.5$). The average ratio of the maximum amount of data transferred in the network in (a), is relative to the corresponding baseline sharing-oblivious task allocation algorithms. The lower segment of the bars represents the data transferred in the case of DSA algorithms while the top hatched segment represents the additional amount of data transferred during the local search phase.

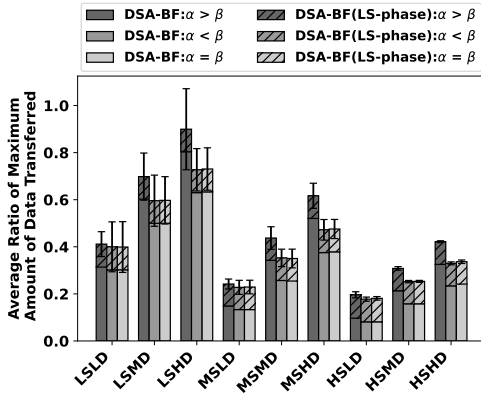
demand. The ratio decreases as the sharing of data items among the tasks increases. For example, in the case of LSLD, DSA-FF leads to 74% less data transferred in the network than FF (corresponding to a ratio of 0.26), while in the case of HSLD, DSA-FF leads to 92.8% less data transferred than FF. A 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.

While the algorithms augmented with local search (DSA-BF-LS, DSA-FF-LS, and DSA-WF-LS) aim to reduce the number of utilized servers and minimize the total amount of data stored on edge servers by co-locating tasks with shared data, they exhibit a small increase in the total amount of data transferred across servers during the Move and Swap operations. Thus, tracking the total amount of data transferred on edge servers is critical for evaluating the trade-off between improved resource utilization and communication overhead. In edge computing systems, data transfer between servers is often costly in terms of latency, energy consumption, and network congestion and an excessive movement of tasks can negate these benefits. To maintain a balance, a small 1% threshold (corresponding to parameter $\gamma = 1.01$) is imposed limiting the allowable amount of data transferred in each local search operation. Since the number of allowed local search operations is also limited by $q_{max} = 10$, the allowed total maximum amount of data transferred in the local search phase is limited to about

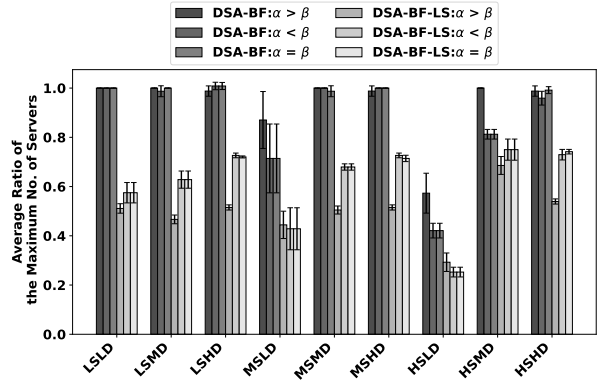
10% of the amount of data transferred in the first phase of the algorithms. This ensures that the algorithms prioritize solutions that are not only efficient in terms of server usage and memory consumption but also practical and scalable in real-world edge systems where bandwidth and latency are limited. As Figure 6a shows, the relative maximum amount of data transferred in the network during the local search phase of the algorithms is within 10% of the amount of data transferred during the first phase of the algorithms.

Figure 6b shows the average amount of data transferred in the network by the proposed data sharing-aware algorithms and the local search phase of 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. This is primarily because high CPU requirements limit the number of tasks that can be placed on a single server. As a result, additional servers must be activated to host the remaining tasks. When tasks that share common data are placed on different servers, the system incurs additional communication overhead, therefore increasing the total amount of data transferred across the network.

The effect of efficiency metric weights (α, β) on DSA-BF and DSA-BF-LS'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 and the number of utilized servers by DSA-BF and DSA-BF-LS. Since the weights are used in DSA-BF but not in the local search phase of DSA-BF-LS, the discussion will mainly focus on DSA-BF, which is the first phase executed by DSA-BF-LS. 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 7a shows an overlay bar plot and the average ratio of the maximum amount of data transferred by DSA-BF and DSA-BF-LS with respect to BF for these three cases. In all nine sharing and demand scenarios DSA-

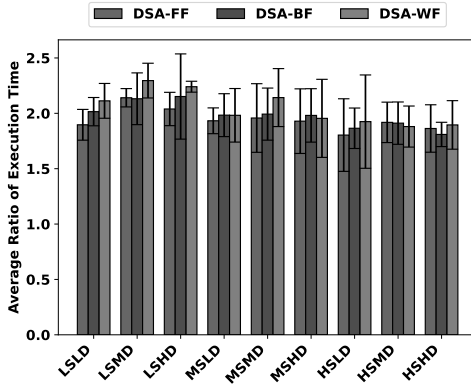


(a) Maximum amount of data transferred relative to baseline algorithms.

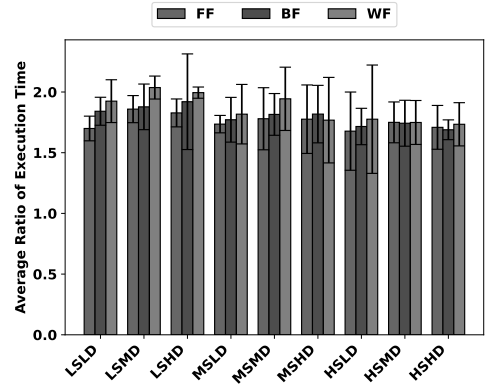


(b) Average ratio of the maximum number of utilized servers.

Fig. 7: The effect of efficiency metric weights (α, β) on DSA-BF and DSA-BF-LS's performance: (a) Maximum amount of data transferred in the network relative to baseline algorithms. The lower segment of the bars represents the data transferred in the case of DSA algorithms, while the top hatched segment represents the additional amount of data transferred during the local search phase. (b) Average ratio of the maximum number of utilized servers.



(a) Data sharing-aware algorithms augmented with local search vs. data sharing-aware algorithms.



(b) Data sharing-aware algorithms augmented with local search vs. baseline sharing-oblivious algorithms.

Fig. 8: Average ratio of execution time: (a) algorithms augmented with local search vs. online data sharing-aware algorithms (DSA), (b) online data sharing-aware algorithms (DSA) vs. online sharing-oblivious algorithms (DSA-BF-LS DSA-WF-LS, DSA-BF, DSA-WF, BF, and WF with $\alpha = \beta = 0.5$).

BF with $\alpha > \beta$ (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. The same analysis applies to DSA-BF-LS. Figure 7b shows the average ratio of the maximum number of utilized servers for both DSA-BF and DSA-BF-LS. For the low sharing cases, DSA-BF obtains almost the same ratio of the maximum number of servers in all three cases considered for α and β . As the sharing increases (MSLD, HSLD, and HSMD cases), DSA-BF obtains the best reduction in the number of servers when $\alpha = \beta$ or $\alpha < \beta$ (cases (ii) and (iii)). Across all three cases

($\alpha > \beta$, $\alpha < \beta$, and $\alpha = \beta$), augmenting DSA-BF with the local search algorithm (DSA-BF-LS) results in a consistent reduction in the average ratio of the maximum number of servers used.

Execution Time. At each task arrival, we record the execution time in millisecond of each algorithms and determined the average among them. In Figure 8a, we determine the ratio of the average execution time of the proposed data sharing-aware algorithms augmented with local search and that of the data sharing-aware algorithms. This ratio varies within the range of [1.80, 2.29] for all three data-sharing-aware online algorithms. Although the maximum ratio for average execution time is 2.29, the maximum execution time of both data sharing-aware algorithms and the algorithms augmented with local search never exceeds 4 milliseconds for one execution (allocating one arriving task). Having a small execution time is very important for an online algorithm, and this is achieved by our proposed algorithms.

In Figure 8b, we determine the ratio of the average execution time of the proposed data sharing-aware algorithms augmented with local search and that of the corresponding

baseline online sharing-oblivious algorithms. This figure shows the ratio for all nine types of instances with different levels of sharing and demand with in a range of [1.67, 2.03] for all three algorithms. Again the maximum execution time of both data sharing-aware algorithms and the local search algorithm never exceeds 4 milliseconds for one execution (allocating an arriving task) making them suitable for online allocation.

Summary of findings. We provide a summary of the experimental results considering the two key metrics, the number of utilized servers and the amount of data transferred in the network. Our online data sharing-aware algorithms are not designed to run concurrently but rather serve as distinct heuristic approaches for task allocation in edge systems where data sharing among tasks plays a crucial role in determining both memory usage and communication overhead. While all six algorithms share the same underlying framework for online task assignment, they differ in the selection policy used when deciding which server to allocate a new task to. Presenting all six algorithms allows for a comparative evaluation under varying CPU demand and data sharing conditions. The results of the comparison summarized below can be used to guide the selection of the most suitable algorithm for specific operating scenarios characterized by the tasks' CPU demand and their data sharing characteristics.

Overall, DSA-FF then DSA-BF consistently deliver the best results minimizing both the number of utilized servers and the data transferred in the network across all nine different scenarios. For low to medium demand with different sharing scenarios, DSA-FF is generally more effective in reducing the number of utilized servers and the total amount of data transferred in the network compared to DSA-BF and DSA-WF. In high demand scenarios (e.g., LSHD, MSHD, HSHD), both DSA-FF and DSA-BF can be used interchangeably while DSA-FF obtains a slightly better performance in terms of the number of utilized servers and the total amount of data transferred. In the case of DSA-BF it is also important to discuss the choice of α and β (i.e., the weight parameters for CPU capacity and memory capacity used in the efficiency function) that are used to fine-tune DSA-BF. In scenarios with high CPU demand tasks, choosing $\alpha > \beta$, where α is large (close to 1) and β is small (close to 0), is a good choice. This places more importance on the CPU capacity in the efficiency metric. In scenarios in which the tasks share a lot of data, $\alpha < \beta$ (with β large, close to 1, and α small, close to 0) or $\alpha = \beta$ would be suitable choices. The algorithms augmented with local search, DSA-FF-LS, DSA-BF-LS, and DSA-WF-LS, obtain the best performance in terms of the number of utilized servers across all scenarios. This is achieved at the expense of a small increase in the data transferred in the network compared to DSA-FF, DSA-BF, and DSA-WF, respectively. DSA-FF-LS transfers the smallest amount of data among all algorithms augmented with local search across all scenarios.

7 CONCLUSION AND FUTURE WORK

Many tasks offloaded to edge servers process the same data items. We take into account the sharing of data among tasks in order to reduce the traffic in the edge network,

the memory usage at the edge servers, and the number of servers needed to execute the tasks. Thus, 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. We also augment these algorithms with a local search procedure that iteratively attempts to improve the solutions obtained by our data sharing-aware algorithms by exploring the neighborhood of the current solution and making minor modifications. We performed an extensive performance analysis by comparing our proposed data sharing-aware online algorithms with several baseline online sharing-oblivious 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. Also, compared to a sharing-aware baseline our proposed algorithms are able to reduce the amount of data transferred in the network by 18.1% to 86.9% and the number of utilized servers by 35.4% to 88%. Although the execution time for the algorithms augmented with local search increases by 80% to 129% compared to the data sharing-aware online algorithms, this is less than 4 millisecond per task allocation in average for all data sharing-aware online algorithms. We provide a summary of the findings that can be used as a guideline for choosing a specific algorithm for a given practical scenario characterized by the tasks' CPU demand and their data sharing characteristics.

In our future work, we plan to deploy our algorithms on an experimental edge computing testbed and investigate their performance. While latency was not the primary optimization objective of this study, incorporating latency-awareness into the data sharing-aware task allocation framework represents an important future direction, particularly for delay-sensitive applications where transmission and response times play a critical role. We also aim to extend the current model by incorporating network topology and bandwidth constraints into the task allocation process, enabling the framework to take into account the link capacities, edge tiers, and potential contention across servers located in different network domains.

ACKNOWLEDGMENT

This paper is a revised and extended version of [8] presented at the IEEE Int. Conf. on Edge Computing (EDGE 2024). This research was supported in part by the US National Science Foundation under Grants CCF-2118202 and CNS-2231523.

REFERENCES

- [1] L. Liu, Z. Dong, Y. Wang, and W. Shi, "Prophet: Realizing a predictable real-time perception pipeline for autonomous vehicles," in *Proc. IEEE Real-Time Systems Symp. (RTSS)*, 2022, pp. 305–317.
- [2] 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.
- [3] G. Dermier, T. Gutekunst, E. Ostrowski, and F. Ruge, "Sharing audio/video applications among heterogeneous platforms," in *Proc. 5th IEEE Int. Workshop on Multimedia Comm.*, 1994, pp. 1–5.
- [4] M. Welsh and G. Mainland, "Programming sensor networks using abstract regions," in *Proc. USENIX Symp. Networked Syst. Design and Impl.*, vol. 4. San Francisco, California, USA, 2004, pp. 3–3.

- [5] H. B. Pasandi and T. Nadeem, "Convince: Collaborative cross-camera video analytics at the edge," in *IEEE Int. Conf. Pervasive Comp. and Comm. Workshops*, 2020, pp. 1–5.
- [6] M. Vahabi, H. Fotouhi et al., "Federated learning at the edge in industrial internet of things: A review," *Sustainable Computing: Informatics and Systems*, p. 101087, 2025.
- [7] J. Chakareski, "Vr/ar immersive communication: Caching, edge computing, and transmission trade-offs," in *Proc. Workshop on Virtual Reality and Augmented Reality Network*, 2017, pp. 36–41.
- [8] S. Rabinia and D. Grosu, "Data sharing-aware online algorithms for task allocation in edge computing systems," in *IEEE Int. Conf. Edge Computing and Comm. (EDGE)*. IEEE, 2024, pp. 139–148.
- [9] 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.*, vol. 6, no. 3, pp. 5853–5863, 2019.
- [10] I. Ullah, H.-K. Lim, Y.-J. Seok, and Y.-H. Han, "Optimizing task offloading and resource allocation in edge-cloud networks: a drl approach," *J. of Cloud Computing*, vol. 12, no. 1, p. 112, 2023.
- [11] Y. Xiao, "A task allocation method in edge computing based on multi-objective optimization," in *Int. Conf. on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*. IEEE, 2022, pp. 247–251.
- [12] Y. Sahni, J. Cao, and L. Yang, "Data-aware task allocation for achieving low latency in collaborative edge computing," *IEEE Internet of Things J.*, vol. 6, no. 2, pp. 3512–3524, 2018.
- [13] Q. Chen, Z. Zheng, C. Hu, D. Wang, and F. Liu, "Data-driven task allocation for multi-task transfer learning on the edge," in *IEEE 39th Int. Conf. on Distrib. Comp. Syst.*, 2019, pp. 1040–1050.
- [14] F. Liu, J. Huang, and X. Wang, "Joint task offloading and resource allocation for device-edge-cloud collaboration with subtask dependencies," *IEEE Trans. on Cloud Computing*, vol. 11, no. 3, pp. 3027–3039, 2023.
- [15] K. Zhu, Z. Zhang, S. Zeadally, and F. Sun, "Learning to optimize workflow scheduling for an edge–cloud computing environment," *IEEE Trans. on Cloud Computing*, vol. 12, no. 3, pp. 897–912, 2024.
- [16] Z. He, Y. Guo, X. Zhai, M. Zhao, W. Zhou, and K. Li, "Joint computation offloading and resource allocation in mobile-edge cloud computing: A two-layer game approach," *IEEE Trans. on Cloud Computing*, vol. 13, no. 1, pp. 411–428, 2025.
- [17] 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.
- [18] 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.
- [19] M. Siew, D. Cai, L. Li, and T. Q. Quek, "Dynamic pricing for resource-quota sharing in multi-access edge computing," *IEEE Trans. Network Sci. and Eng.*, vol. 7, no. 4, pp. 2901–2912, 2020.
- [20] Y. Zhao, D. Guo, J. Xu, P. Lv, T. Chen, and J. Yin, "Cats: Cooperative allocation of tasks and scheduling of sampling intervals for maximizing data sharing in wsns," *ACM Trans. on Sensor Networks (TOSN)*, vol. 12, no. 4, pp. 1–26, 2016.
- [21] 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.
- [22] S. Rabinia, N. Didar, M. Brocanelli, and D. Grosu, "Algorithms for Data Sharing-Aware Task Allocation in Edge Computing Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 36, no. 1, pp. 15–28, January 2025.
- [23] 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.
- [24] 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.
- [25] —, "Sharing-aware online virtual machine packing in heterogeneous resource clouds," *IEEE Trans. on Parallel and Distributed Systems*, vol. 28, no. 7, pp. 2046–2059, 2016.
- [26] K. Katsalis, T. G. Papaioannou, N. Nikaein, and L. Tassioulas, "Sla-driven vm scheduling in mobile edge computing," in *IEEE 9th Int. Conf. on Cloud Computing*, 2016, pp. 750–757.
- [27] R. Mahmud, K. Ramamohanarao, and R. Buyya, "Latency-aware application module management for fog computing environments," *ACM Trans. Internet Technol.*, vol. 19, no. 1, 2018.
- [28] 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.
- [29] 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 Trans. on Communications*, vol. 67, no. 1, pp. 489–502, 2019.
- [30] P. Erdős and A. Rényi, "On random graphs i," *Publicationes Mathematicae Debrecen*, vol. 6, p. 290, 1959.
- [31] N. Didar and M. Brocanelli, "Joint ai task allocation and virtual object quality manipulation for improved mar app performance," in *Proc. IEEE 44th Int. Conf. on Distributed Computing Systems (ICDCS)*, 2024, pp. 1154–1165.

BIOGRAPHIES



Sanaz Rabinia received her Ph.D. in Computer Science from Wayne State University in 2025. She earned her M.Sc. in Mathematics from Sharif University of Technology (Iran) in 2013 and her B.Sc. in Mathematics from Shahid Beheshti University (Iran) in 2011. Her research interests include edge computing, cloud computing, parameterized algorithms, approximation algorithms, parallel computing, and combinatorial optimization. She received an Honorable Mention for the 2024 NCWIT Collegiate Award and the 2025 Ralph H. Kummier Distinguished Achievement Award in Graduate Student Research from the Wayne State University College of Engineering. She is a student member of the ACM and the IEEE.



Daniel Grosu received the Diploma in engineering (automatic control and industrial informatics) from the Technical University of Iași, Romania, in 1994 and the MSc and PhD degrees in computer science from the University of Texas at San Antonio in 2002 and 2003, respectively. Currently, he is a professor in the Department of Computer Science, Wayne State University, Detroit. His research interests include parallel and distributed computing, graph algorithms, approximation algorithms, and topics at the border of computer science, game theory and economics. He has published more than one hundred peer-reviewed papers in the above areas and is an *IEEE Computer Society Distinguished Contributor*. He serves as a senior associate editor for *ACM Computing Surveys* and as an associate editor for *IEEE Transactions on Parallel and Distributed Systems*, and *IEEE Transactions on Cloud Computing*. He is a Distinguished Member of the ACM, a senior member of the IEEE, and the IEEE Computer Society.