IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 13 November 2022, accepted 27 November 2022, date of publication 6 December 2022, date of current version 14 December 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3227210

RESEARCH ARTICLE

Selfish Routing Game-Based Multi-Resource Allocation and Fair Scheduling of Indivisible Jobs in Edge Environments

HAJAR SIAR AND MOHAMMAD IZADI
Department of Computer Engineering, Sharif University of Technology, Tehran 11155-9517, Iran
Corresponding author: Mohammad Izadi (izadi@sharif.edu)

ABSTRACT Distributed and heterogeneous edge computing environments require efficient allocation and scheduling of multiple users' applications. This paper presents a game-theoretic solution to model the competition between time-sensitive internet of things (IoT) applications with indivisible loads to be allocated and scheduled to edge servers. We model the allocation problem as a selfish routing game such that no job is unsatisfied with its allocation. Also, the allocated jobs are scheduled using a weighted time-sharing approach, which assigns resources to jobs proportional to their demands and deadlines. We proved the existence of pure Nash equilibrium in the introduced game and presented a greedy and polynomial time algorithm to obtain the solution called SAFSA. Using an extended version of the CloudSim simulator, we assess our approach in different situations by employing a real-world dataset of the Google cluster. Since the proposed algorithm considers jobs' deadlines beside resource requests in allocation and scheduling decisions, it can simultaneously minimize average response time and maximize the number of jobs completed within their deadlines.The simulation results confirm the ability of SAFSA to efficiently execute the jobs within their deadlines, especially when there are many IoT devices and edge servers in the system, which is the main characteristic of IoT environments.

INDEX TERMS Edge computing, job allocation, Nash equilibrium, selfish games, time-sharing scheduling.

I. INTRODUCTION

The heterogeneous environment of edge computing has the potential to execute the distributed and real-time applications of IoT efficiently by offloading them to remote edge servers [1], [2], [3]. There is heterogeneity in the environment and applications, such as different processing and networking resources and different resource requirements for the applications.

Even though edge computing can handle the diverse demands of IoT applications, receiving multiple applications with varying demands that insist on a particular level of performance may prevent efficient achievement for all of them. In this case, meeting the requirements of one application may stop others from benefiting [4]. It becomes more challenging when QoS requirements, such as the deadline, are involved. Consequently, allocating resources to applications in proportion to their demands is crucial to ensure that all of them are satisfied with their performances and QoS requirements.

Depending on whether an application's load can be arbitrarily divisible, applications are classified into two categories: divisible and indivisible loads. Typically, fair resource allocation research focuses on jobs with divisible loads, although fractional loads may not be acceptable in real-world applications [5], [6], [7], [8]. Therefore, this paper focuses on the fair allocation and scheduling of time-sensitive jobs with indivisible loads. The limited resources of edge servers make it impossible to execute all jobs within their deadlines. So, our mechanism attempts to reduce the average response time and the number of jobs that are not completed by the deadline.

In this paper, we address the following two main problems:

- Selecting the appropriate edge server for each job (where): The allocation decisions of the jobs affect the performance of each other. Therefore, we developed a

The associate editor coordinating the review of this manuscript and approving it for publication was Mehdi Sookhak.

selfish allocation game. As part of this game model, each job seeks to select an edge server that minimizes its response time. To reduce the number of jobs violating their deadlines, we considered jobs' deadlines along with resource requirements when estimating response times.

- Scheduling the jobs allocated to each edge server (when): Our scheduling approach addresses two issues. The first issue is fair resource allocation proportional to jobs' demands and deadlines, and the second is the lack of starvation for low-priority jobs. We addressed the first issue by assigning weights corresponding to the resource demands and the deadline of the job to specify its share of each resource. On the other hand, in most scheduling approaches, like the non-preemptive ones, low-priority jobs wait until the execution of those with higher priority in the scheduling queue is completed. It can cause severe starvation for jobs with lower priority and cause their deadlines to be missed, especially in environments with many jobs, such as IoT. Time-sharing scheduling allocates resources to jobs, in turn preventing starvation. Thus, we used weighted time-sharing scheduling [9] to execute the jobs proportionally to their resource demands and deadlines, keeping starvation of low-priority jobs.

We are considering fairness in assigning resources to jobs in proportion to their demands, such that all of them are satisfied with their response times. Due to the egoistic behavior of jobs in achieving better performance [10], [11], [12], we designed a selfish job allocation game. In our game model, edge servers execute jobs according to a weighted time-sharing scheduling algorithm, which allocates resources fairly between jobs. We proved the existence of pure Nash equilibrium in the introduced game and then designed an algorithm to reach it. Following is a summary of the significant contributions of this paper.

- We consider the selfish allocation of indivisible jobs to multi-resource edge servers to minimize response times and fairly allocate resources proportional to jobs' resource demands and deadlines. We model this problem as a selfish game between users' jobs in which players attempt to minimize their response times. We believe this is the first work to deal with the joint fair allocation and scheduling of indivisible and time-sensitive jobs using a game-theoretic approach when edge servers have multiple resources.
- We prove the existence of pure Nash equilibrium in the defined selfish allocation and fair scheduling game. Then present a greedy and polynomial time algorithm to obtain the solution called SAFSA.
- We analyze the performance of our proposed approach using the real-world dataset of Google cluster traces and a simulation toolkit based on CloudSim [13], [14]. The evaluation results indicate that our joint allocation and scheduling game significantly minimizes the average response time and deadline miss rate of jobs.

This paper follows the following structure. Section II demonstrates a real-world scenario as a motivation for this study. Section III presents related works on fair allocation and scheduling. The system model is explained in section IV. We describe our weighted time-sharing scheduling and selfish allocation game in sections V and VI, respectively. Then we present our algorithm in Section VII. Section VIII contains the evaluation settings and the experimental analysis. The paper is concluded in section IX.

II. MOTIVATION

In vehicular edge computing (VEC), many IoT devices in different places, such as roads, vehicles, drivers, and passengers, continuously generate data. These data bring enormous delay-sensitive and computation-sensitive applications, including intelligent navigation, autonomous driving, intelligent entertainment, accident warnings, augmented reality-supported gaming, AI-based pedestrian detection, and fuel scheduling [15], [16]. The data generated by these applications must be processed and stored on time, which is not in the ability of resource-limited IoT devices. VEC uses roadside units (RSUs) equipped with edge servers to offload and execute jobs sent from IoT devices, helping them to be executed on time. The applications have a specific data size which must be sent to RSUs through communication links. Also, they need a specific amount of computation resources [15], [17]. All of these applications are time-sensitive, and a predefined deadline can be considered for the maximum response time of the jobs. However, their sensitivity to being completed within their deadline is varied. Some applications' response time must not exceed their deadlines, and their deadline violation has catastrophic consequences. These applications have a hard deadline, like driver assistance in VEC to prevent accidents. On the other hand, some applications' deadlines are considered soft and less safety-critical, and their missing is sometimes acceptable. Applications such as intelligent entertainment or indoor positioning systems in vehicles are examples of applications with a soft deadline. In these applications, it is attempted to complete as many jobs as possible within their deadlines [18], [19], [20]. This paper focuses on applications with soft-deadline.

Due to the delay sensitivity of applications and limited processing resources in edge servers, offloading the IoT devices' soft-deadline jobs to arbitrary RSUs, such as the closest unit, is not an efficient solution, especially when data load is very high because of critical situations like traffic or accidents. In this case, some RSUs may be overloaded and cannot schedule and execute their offloaded jobs on time, so the deadline of most of the soft-deadline jobs may be missed. Since these jobs are from different IoT devices, there is competition between jobs to receive their output in a minimum time, satisfying their deadlines. So, the current study helps VEC to fairly offload and schedule soft-deadline applications' jobs (e.g., intelligent entertainment or indoor positioning systems in vehicles) among RSUs to minimize the average response time and the number of jobs with a missed deadline.

III. RELATED WORKS

Fair allocation in multi-resource environments is first introduced with the economic mechanism of dominant resource fairness (DRF) by Ghodsi [6]. The DRF is based on fairness for users' utilization, considering all server resources are pooled together into a single resource, which is incorrect for distributed computing environments, such as cloud and edge computing. Many studies are tried to improve this method by reducing the single resource server assumption [7], [21]. Meskar [8] demonstrated that DRF might not provide a fair allocation mechanism for edge computing environments, where both computation and networking resources are variable. Then, they proposed a fair allocation mechanism for these environments. Max-min fairness for jobs with multiple parallel tasks in the scheduling of distributed computing platforms is studied in [22]. In this study, the utility of each job is a decreasing function of its completion time. The authors first present two notions of approximation in max-min fairness of multi-task jobs, then scheduling is introduced considering these notations. The problem of fair allocating multiple resources in multiple servers among users in mobile edge computing is studied in [23]. This paper considers that each user cannot establish a wireless connection to multiple edge servers, and her/his jobs are executed on exactly one edge server, also Leontief utilities are assumed for the users. It introduces a mechanism called MAGIKS to reach the fair Nash equilibrium solution. It is a polynomial time mechanism, provided that resource demands are discrete.

Data centers consist of multiple heterogeneous resources, and fair allocation is challenging. In [24], Poullie et al. introduced a survey on fair allocation in data centers. Besides presenting a definition of fairness in multi-resource allocation and relevant state-of-the-art, the authors explained utility functions and allocation characteristics to support fair data center resource allocation. The study of [25] presents multi-user task scheduling among fog nodes with two priority queues. Tasks can be scheduled in a fog node that receives offloaded tasks or in a neighboring fog node. In this study, fairness is defined as a stable allocation of tasks among scheduling queues, not the satisfaction of tasks. The paper of [26] tries to establish fairness among offloading services of resource-limited ground nodes (GNs), considering the trade-off of energy consumption between computing and caching operations. It attempts to assign UAV's computation resources among GNs equivalently. Furthermore, there are studies considering the fairness of an allocation solution regarding servers' utilization [27], [28], [29].

Round-robin [30] is a fair scheduling algorithm that assigns resources to jobs in equal time-slices in turn. Although this simple algorithm tries to allocate equal resources to all jobs, it is not efficient when there are many jobs with predefined QoS requirements like the deadline [31]. Many studies have investigated extending the round-robin algorithm to consider the priority of the jobs [32], [33], [34]. Stoica, et al. proposed a weighted round-robin algorithm in [35] for scheduling real-time processes in time-sharing systems. Also, a weighted round-robin algorithm for packet scheduling is proposed in [36].

The study of [31] also introduced a learning-based approach to allocate and schedule the jobs among edge servers fairly, considering the dynamic nature of network conditions and server loads. This solution introduces a weighted round-robin scheduling algorithm combined with deep reinforcement learning to schedule the jobs within the edge servers.

There are research studies on job allocation and scheduling in edge and cloud infrastructures, considering QoS requirements of users such as the deadline, not addressing a fair solution [37], [38], [39]. Liu, et al. studied task scheduling in a mobile cloud computing environment to minimize the application's total execution cost and load based on a novel scheduling algorithm called HEFT-T [40]. The authors considered interdependent tasks and two cases of unconstrained and deadline-constrained tasks. The paper of [41] introduced a game-theoretic solution for multi-user computation offloading in a three-tier mobile cloud computing environment, attempting to find the Nash equilibrium strategy for mobile users' offloading. Yuqing Li, et al. studied cooperative service placement and scheduling in edge clouds in [42]. This solution reduces cost by exploiting spatial-temporal diversities in workload and resource cost among federated edge clouds. The authors investigate this problem in two steps. This problem is formulated first without considering deadlines, and then tasks' deadlines are added to the problem, and a cooperative solution is presented. The paper of [43] presented an approximated solution for the computation offloading of delay-sensitive tasks in mobile edge computing to maximize energy saving. The paper of [44] introduced a three-layer task offloading and allocation framework in a mobile computing environment for multiple tasks of a single application, taking into account the task dependencies and users' mobility, to reduce the processing delay.

There is a significant difference between the studies presented and ours. The differences between our work and [8] are as follows. First, our approach attempts to reach a game-theoretic solution using the Nash equilibrium such that no player is dissatisfied with her/his job's response time. Second, we introduced a joint allocation and scheduling solution in multi-resource environments of edge computing. Third, we consider jobs' deadlines in the scheduling and try to minimize the deadline miss rate of jobs. In the MAGIKS mechanism [23], the assumption of executing each user's job at exactly one edge server is similar to that considered in the current study. In contrast to our study, the edge servers in MAGIKS are homogeneous. Also, it did not consider any QoS constraint (e.g., deadline) for the jobs. Most previous works on single/multi-resource fair allocation [6], [7], [8], [21] consider jobs to be divisible, whereas our study considers indivisible loads. In addition, the paper of [41] differs from ours in some directions. Our solution addresses joint allocation and scheduling problems considering the fair assignment of jobs with QoS constraint of the deadline. However, the

paper of [41] models the response time of applications using queueing theory without considering any QoS constraint. In addition, there are differences between our work and [31]. Our joint allocation and scheduling game-theoretic solution outputs a pure Nash equilibrium where no player (i.e., job) desires to change its allocation strategy, while [31] is a learning-based technique.

We did not find in the literature any game-theoretic approach presenting a fair joint allocation and scheduling solution for time-sensitive jobs with indivisible loads in a multi-resource edge computing environment.

IV. SYSTEM MODEL

Figure 1 shows an overall view of the considered system model. It consists of n IoT devices and m edge servers. Each IoT device belongs to a user and receives/sends data from/to its surrounding environment. The IoT device's input data triggers an IoT application associated with its specified user. We assume the applications as independent jobs with indivisible loads that are time-sensitive. So, two terms, IoT application and job, are the same throughout this paper. The set of input jobs to the system in each time is represented by $\tau = \{T_1, T_2, \ldots, T_n\}$, and T_i represents the job associated with user i. There are m access points with different wireless link bandwidths. Each access point is associated with a server with k type of computational resources at the network's edge. We consider every access point and associated server an edge server with a total of $k+1$ resources, where link bandwidth is its $k+1$th resource. So, we assume the set of edge servers as $E = \{R_1, R_2, \ldots, R_m\}$. Each edge server R_j is represented as $(R_{j_{c_{c=1:k}}}, R_{j_b})$, where R_{j_c} indicates the capacity of the computational resource of type c in the server and R_{j_b} represents the link capacity for the jth access point. So, we assume each job demands k types of computational resources besides the link bandwidth as $k+1$th resource. We are representing every job T_i with a tuple $(T_{i_{c_{c=1:k}}}, T_{i_b}; d_i)$ where T_{i_c} represents T_i's demand of computational resource of type c, and T_{i_b} is T_i's link request or data size which should be transmitted from the wireless communication link. We assume the resource demands of all jobs for each resource type are non-zero: $T_{i_{c_{c=1:k}}} > 0$, and $T_{i_b} > 0$; $\forall i$. Every job T_i announces a deadline d_i, and desires to be executed within this time.

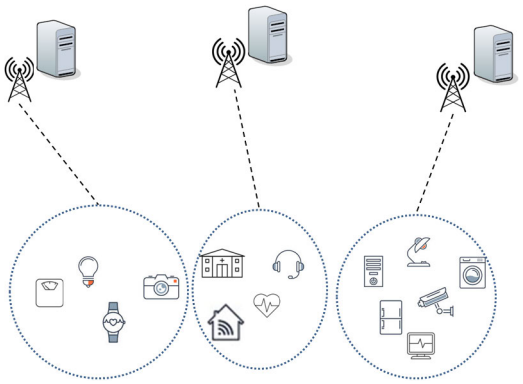


FIGURE 1. System model.

It should be noted that we used the capacity of the links in bit rate, not the actual physical channel's bandwidth (Hz). Considering the actual physical bandwidth of edge servers' links in Hz will result in different values for the equations of this paper. Using the following Shannon's theory [45], [46], links' capacity in bit-rate calculates from physical bandwidth (Hz).

$$C = B \log_2(1 + \frac{S}{N}) \tag{1}$$

where, S is the average signal power and N is the average noise power.

Each edge server executes the jobs of a set of IoT devices. The assumption is that jobs allocated to one edge server are not transferred to another. Every job desires to be executed by its deadline while minimizing response time. However, this can not be obtained for all jobs due to the limitations in edge servers' resources and different resource requirements and deadlines. Thus, our resource allocation and scheduling solution seeks to minimize the average response time of input jobs and the number of jobs that miss their deadline. A list of all notations used in this paper is presented in table 1.

V. WEIGHTED TIME-SHARING SCHEDULING

The process of transmitting and executing the jobs in the system is as follows. Every job T_j transfers from its associated IoT device to an edge server, executing the job according to its resource requirements. Then the processing output of the job is sent back to that device. The output data size is in bytes, and very small compared to the job's size, which is ignorable [31], [47], [48]. Therefore, we consider the response time of a job as the sum of two values: the transmission time, which is the time of sending the job from the IoT device to the edge server, and the computation time, which is the time of processing the job in k computation resources of the edge server. So, a job is executable if transmitted through the wireless communication link, and its data is placed on the allocated edge server.

Considering the competition among users' jobs to minimize their response times, we formulated the joint allocation and scheduling problem as a selfish game among jobs. Non-preemptive scheduling methods lead to starvation for jobs with low-priority, which likely miss their deadlines [31]. So, these scheduling approaches are not fair in assigning resources to jobs. Hence, we used a weighted time-sharing scheduling method to allocate resources to jobs proportional to their demands and deadlines, using $k+1$ weight parameters as their share of resources. Our selfish allocation and fair scheduling solution attempt to yield a strategy profile of the game in which all users' jobs are satisfied with their response times.

Jobs determine one value for their deadlines and desire the sum of transmission time in the communication link, and the execution time in computation resources does not violate this value. So we split each job's deadline into transmission

TABLE 1. List of notations.

Notation	Definition
n	number of users and their corresponding IoT devices
m	number of edge servers
i	index of jobs
j	index of edge servers
T_i	associated job to user i which triggers from its corresponding IoT device
R_j	edge server j
R_{j_c}	the capacity of computation resource of type c for R_j
R_{j_b}	link capacity for R_j
T_{i_c}	T_i's request for computation resource of type c
T_{i_b}	T_i's link request
d_i	T_i's deadline
d_{i,j_c}	computation deadline of job T_i if allocated to R_j
d_{i,j_b}	transmission deadline of job T_i if allocated to R_j
w_{i,j_κ}	job T_i's weight on resource κ of edge server R_j
f_{i,j_k}	job T_i's share of resource κ of edge server R_j
x_i	allocation strategy of player i (i.e. job T_i)
$\boldsymbol{X}$	Strategy profile of the game
C^i	response time of job T_i
$\Psi_{x_{i_c}}$	the sum of weights of jobs in resource c of edge server x_i
$\Psi_{x_{i_b}}$	the sum of weights of jobs in the link resource of edge server x_i
cost^{R_j}	the cost of edge server R_j in the selfish allocation game

and computation deadlines for executing the job in the edge servers' resources. In more detail, if job T_i is allocated to edge server R_j, its transmission and computation deadlines are represented by d_{i,j_b}, and d_{i,j_c}, respectively. All computation resources of R_j, suppose d_{i,j_c} for their deadlines, which is obtained using the maximum ratio of request to the resource capacity among computation resources. We can estimate these parameters as follows.

$$\begin{cases} d_{i,j_c} + d_{i,j_b} = d_i \\ \dfrac{(\max_{c\in[1:k]}\{T_{i_c}/R_{j_c}\})}{(T_{i_b}/R_{j_b})} = \dfrac{d_{i,j_c}}{d_{i,j_b}} \end{cases} \tag{2}$$

Our weighted time-sharing scheduling estimates weights for jobs to determine their shares of the edge server's resources. If w_{i,j_κ} is job T_i's weight in resource κ of edge server R_j, bellow equation evaluates this job's share of this resource [35].

$$f_{i,j_k} = \frac{w_{i,j_\kappa}}{\sum_{l\in I_{j_\kappa}} w_{l,j_\kappa}} \tag{3}$$

where I_{j_κ} is the set of jobs in the scheduling queue of resource κ of edge server R_j.

Computation resources can process at most one job at a time, so these resources are assigned to every job successively, corresponding to the job's share. On the other hand, edge server R_j's link bandwidth divides into χ separate parts (for example, different frequency bands in the frequency division multiple access (FDMA) or different time slots in the time division multiple access (TDMA)), where χ is the number of jobs allocated to R_j. The share of link bandwidth for job T_i is proportional to its demand and deadline. Although multiple access methods can reduce wireless interference to some extent, the interference between edge servers is still established. Please note that we refer to these examples to provide intuition on the function of the introduced scheduling method. So, the challenges related to interference management are not in the scope of this article. However, there are several studies in the literature [49], [50], [51], [52] to address these challenges, that can be used in the current study.

We estimate the weight of every job T_i on resource κ of edge server R_j according to equation (4).

$$\frac{w_{i,j_\kappa}}{\sum_{l\in I_{j_\kappa}} w_{l,j_\kappa}} \times d_{i,j_\kappa} = \frac{T_{i_\kappa}}{R_{j_\kappa}} \tag{4}$$

where, d_{i,j_κ} is the deadline of job T_i on resource κ of edge server R_j, T_{i_κ} is job T_i's request for this resource, and R_{j_κ} is the capacity of resource of type κ in edge server R_j. So, we can estimate every w_{i,j_κ} as follows.

$$w_{i,j_\kappa} = \frac{\frac{(T_{i_\kappa}/R_{j_\kappa})}{d_{i,j_\kappa}} \sum_{l\in I_{j_\kappa},\, l\neq i} w_{l,j_\kappa}}{1 - \frac{(T_{i_\kappa}/R_{j_\kappa})}{d_{i,j_\kappa}}} \tag{5}$$

Our weighted time-sharing scheduling assigns resources to jobs according to their shares, estimated using equation (3). Therefore if R_{x_i} is the allocated edge server to job T_i, and $\boldsymbol{X} = (x_1, \ldots, x_i, \ldots, x_n)$ is the allocation decision for input jobs, the below equation estimates the response time of T_i.

$$C^i(\boldsymbol{X}) = \left(\max_{c=1:k}\left(\Psi_{x_{i_c}} \times \frac{T_{i_c}}{R_{x_{i_c}} \times w_{i,x_{i_c}}}\right) + \Psi_{x_{ib}} \times \frac{T_{i_b}}{R_{x_{i_b}} \times w_{i,x_{i_b}}}\right) \tag{6}$$

where $\Psi_{x_{i_c}}$ is the sum of weights of jobs in resource c of edge server R_{x_i}, and $\Psi_{x_{i_b}}$ is the sum of weights of jobs in the link bandwidth of R_{x_i}.

Hence, we have formulated the joint fair allocation and scheduling problem to minimize jobs' response time in the introduced system model. According to the illustrated scheduling, in each edge server R_j, the shares and response time of every job T_i depends on the number of jobs assigned to that edge server and their demands and deadlines. As a result, we model the joint allocation and scheduling problem as a selfish game, which we will explain in the next section.

VI. SELFISH JOB ALLOCATION GAME

According to the defined system model, we want to allocate users' jobs to edge servers such that resources are assigned

proportional to jobs' demands and deadlines, and no job is unsatisfied with its response time. Every job attempts toward its benefit without caring for others. Therefore, we are facing a selfish allocation game among n users' jobs as players of the game [53], [54], [55]. We modeled this strategic problem as a routing game. The routing game is defined among players who want to transfer their loads in a network, including links with specified capacities. Every link has a cost, a function of the amount of traffic that uses the link. Selfish players choose routes to minimize the cost incurred. In equilibrium, all players choose a path of minimum cost [55], [56].

Since the jobs have indivisible loads, every job is assigned to exactly one edge server. So, the players' strategy is choosing one edge server among m edge servers (i.e., pure strategy). We used a variable $x_i \in [1, m]$ to represent the strategy of player i in our selfish game; it denotes the index of the edge server selected by job T_i. Therefore, the strategy profile of our routing game can be shown using vector $\boldsymbol{X} = (x_1, \ldots, x_i, \ldots, x_n)$, where every entry x_i of this vector represents the allocation decision of player i.

Considering the cost of job T_i in selecting an edge server, as the cost of all jobs selecting that edge server, we define each edge server R_j's cost as follows.

$$\text{cost}^{R_j}(\boldsymbol{X}) = \Big(\max_{c=1:k}\Big(\Psi_{R_{j_c}} \times \sum_{i:x_i=j} \frac{T_{i_c}}{R_{j_c} \times w_{i,j_c}}\Big) + \Psi_{R_{j_b}} \times \sum_{i:x_i=j} \frac{T_{i_b}}{R_{j_b} \times w_{i,j_b}}\Big) \tag{7}$$

According to equations (6) and (7), minimizing the cost of each edge server is obtained by reducing the cost of every job assigned to that edge server.

In the defined selfish routing game, each player attempts to minimize her/his response time. The most common concept in solving such games is the Nash equilibrium [53], [55]. The following claim explains when strategy profile $\boldsymbol{X}$ is a Nash equilibrium of the defined game.

Claim 1: $\boldsymbol{X} = (x_1, \ldots, x_i, \ldots, x_n)$ *represents a Nash equilibrium in our selfish allocation and fair scheduling game if we have the following for all jobs* T_i.

$$C^i(x_i, x_{-i}) < C^i(x_i', x_{-i}) \tag{8}$$

where x_{-i} represents the allocation strategy of players other than i. So, in the Nash equilibrium, no player can decrease her/his response time by changing her/his allocation.

In general, Nash equilibria are defined in terms of probability distributions over the set of pure strategies called mixed strategies. Since we considered indivisible jobs which must be assigned to specific edge servers, we are interested only in pure strategy Nash equilibrium [28], [53].

Theorem 1: At least one pure Nash equilibrium exists for the proposed selfish allocation and fair scheduling game.

To prove theorem 1, we define lexicographically minimum vectors as follows.

Definition 1 (Lexicographically Minimum Vector): For any two $m \times 1$ vectors $\mathbf{x}$ and $\mathbf{y}$; $\mathbf{x}$ is *lexicographically less than* $\mathbf{y}$, if there is an index $k \leq m$ such that for each index $i \leq k, x_i = y_i$, while $x_k < y_k$. The lexicographically minimum vector is the least element of this total order.

Having definition 1, we can prove theorem 1 [56].

Proof: Every pure strategy profile $\boldsymbol{X}$ induces a sorted cost vector $\boldsymbol{\zeta} =< \text{cost}^{R_1}, \text{cost}^{R_2}, \ldots, \text{cost}^{R_m} >$, in non-increasing order such that $\text{cost}^{R_1} \geq \text{cost}^{R_2} \geq \ldots \geq \text{cost}^{R_m}$. We show that the pure Nash equilibrium strategy profile of $\boldsymbol{X}^*$ corresponds to the lexicographically minimum sorted cost vector of $\boldsymbol{\zeta}^*$.

To reach a contradiction, suppose that $\boldsymbol{X}^*$ is not a Nash equilibrium. So, there exists a job $T_{i_{i\in[1:n]}}$ which assigned its load on edge server R_j, (i.e. $x_i = j$) and an edge server $R_{j'_{j'\in[1:m]}}$, such that $\text{cost}^{R_{j'}} < \text{cost}^{R_j}$, if $x_i = j'$. Hence, if job T_i changes her strategy from edge server R_j to $R_{j'}$, the cost of edge server $R_{j'}$ after the change is also smaller than R_j's cost in $\boldsymbol{X}$. Therefore, the sorted cost vector after job T_i changing its strategy from edge server R_j to $R_{j'}$ is lexicographically less than $\boldsymbol{\zeta}^*$, which is a contradiction. □

According to proof of theorem 1, the lexicographically minimum cost vector of edge servers corresponds to a pure Nash equilibrium of the introduced game. Since jobs' demands of resources are non-zero, they must select edge servers to minimize their response times to reach a pure Nash equilibrium. This reduces the sum of the response times of all jobs selecting that edge server (i.e., the cost of the edge server). Therefore, in the pure Nash equilibrium of our game, every job must select an edge server such that its response time is minimized and it has no incentive to change the selected strategy. The following section presents a greedy and polynomial time algorithm to reach the pure Nash equilibrium of the introduced selfish allocation and fair scheduling game.

VII. ALGORITHMIC SOLUTION

In this section, we present a greedy and polynomial time algorithm called SAFSA (selfish allocation and fair scheduling algorithm) to compute a pure Nash equilibrium strategy profile of the introduced game.

SAFSA assigns jobs to edge servers greedy in terms of resource demands and deadlines, algorithm 1 represents its pseudo-code. Inputs of the algorithm represented in the first line, including the set of input jobs of $\{T\}$, the set of available edge servers of $\{E\}$, Ψ_{j_b} as the sum of weights of jobs assigned to the link bandwidth of every edge server R_j, and Ψ_{j_c} is the sum of weights of jobs assigned to each computation resource c of every edge server R_j. Initially, SAFSA assigns every input job to an edge server where the sum of the ratio of resource demand to the resource capacity (i.e., $\frac{T_{i_\kappa}}{R_{j_\kappa}}$) for all resource types of κ is minimized. So, the weights of jobs do not consider in the initial assignment (lines 2 to 4). Also, the cost of edge servers' resources initializes to zero (lines 5-7). After the initial assignment of jobs,

the following process (i.e., lines 9-20) will be done on each edge server. Edge server R_j gives the highest priority to job T_1 if the sum of $\frac{T_{1_{\kappa=1:k+1}}}{R_{j_{K\kappa=1:k+1}} \times d_{1,j_{K\kappa=1:k+1}}}$ is greater than that of jobs with R_j as their strategy (line 10). The reason is that in the introduced weighted time-sharing scheduling, the weight of job T_i in each resource $\kappa \in [1 : k+1]$ of edge server R_j is directly related to the ratio of $\frac{T_{i_K}}{R_{j_K} \times d_{i,j_K}}$ (according to equation (5)). If no job is allocated to R_j, then T_1 will assign to this edge server with weight one, and the sum of weights of resources will be updated (lines 12-13). Otherwise, the sum of weights will be updated after calculating the weight of job T_1 according to equation (5) (lines 15-16). After assigning job T_1 to R_j, the execution time of this job on each resource of R_j will be added to the cost of that resource. Finally, T_1 will be removed from the set of input jobs (i.e., set $\{T\}$) (lines 18-19).

In the following, edge server R_j will be selected for the allocation strategy of every job T_i in the set of unassigned input jobs of $\{T\}$ if the cost of this server is minimized by adding the ratio of $\frac{T_{i_K}}{R_{j_K} \times w'_{i,j_K}} \times \Psi_{j_K}$ to the current cost of every resource κ of R_j. Where w'_{i,j_K} is a temporary weight of T_i in resource $\kappa \in [1 : k+1]$ of R_j by having the current sum of weights in the resources (lines 21-23). Then, SAFSA reruns the process of selecting the job with the highest priority in every edge server (lines 9-20). The algorithm repeats until all jobs are assigned to the edge servers (lines 8-24).

Claim 2: SAFSA computes the pure Nash equilibrium of the introduced selfish allocation and fair scheduling game in a polynomial time.

Proof: We assume $X(s) = (x_1^s, x_2^s, \ldots, x_s^s)$ is the output strategy profile of SAFSA for the first s jobs with the highest priority, and $C^i(X(s))$ is the response time of job T_i in $X(s)$. We represent the allocation strategy of T_i in $X(s)$ with x_i^s, and x_{-i}^s is the allocation strategy of jobs other than T_i.

Since the initial assignment is based on jobs' resource demands and edge servers' available resources, the claim is correct if there is one job in the system. We inductively consider that for all $s \geq 1$, $s \in [1 : n]$, $X(s)$ is a pure Nash equilibrium for the first s jobs with the highest priority. Then we show that $X(s+1)$ is a pure Nash equilibrium for the jobs in $[1 : s+1]$.

Job T_{s+1} will be assigned to edge server R_j where its response time is minimized considering the available resources of the edge servers and the jobs allocated to them. Also, T_{s+1}'s weight in R_j's resources is more than all jobs selected that edge server as their strategy. So T_{s+1} cannot decrease its response time by changing its allocation strategy, and we have the following inequality:

$$C^{s+1}(x_{s+1}^{s+1}, x_{-(s+1)}^{s+1}) < C^{s+1}(x\prime_{s+1}^{s+1}, x_{-(s+1)}^{s+1}) \tag{9}$$

Also, for each job $T_l \neq T_{s+1}$ assigning to edge server $R_q \neq R_j$, we have the following:

$$C^l(X(s+1)) = C^l(x_l^s, x_{-l}^s) < C^l(x\prime_l^s, x_{-l}^s) \tag{10}$$

TABLE 2. Experimental parameters.

parameter	value
Number of IoT devices	50
Number of edge servers	5
Processing power of edge servers	{800 MIPS, 1500 MIPS, 2000 MIPS, 3300 MIPS }
Link capacity of edge servers	{1 GB/s, 5 GB/s, 10 GB/s}

Since T_l had been assigned to an edge server other than R_j, inserting T_{s+1} into the input jobs does not affect its response time, and the equality is correct. Also, the inequality holds because $X(s)$ is assumed to be the Nash equilibrium of the game. On the other hand, the following inequality is correct for each job $T_m \neq T_{s+1}$ that is assigned to R_j.

$$C^m(x_m^{s+1}, x_{-m}^{s+1}) < C^m(x\prime_m^{s+1}, x_{-m}^{s+1}) \tag{11}$$

Since the jobs are assigned to edge servers in order of priority, the priority of T_m in R_j is higher than T_{s+1}, and T_m has no incentive to change its allocation. Therefore, the players (i.e., jobs) have no incentive to change their allocation, and $\boldsymbol{X} = (x_1, x_2, \ldots, x_n)$ is the pure Nash equilibrium of the introduced selfish allocation and fair scheduling game.

The main operation of determining a job with the highest priority for each edge server is in the loop of lines 8-24 of algorithm 1. It is in the order of $O(nmk)$, where n is the number of input jobs, m is the number of edge servers, and k is the number of resources in each edge server. Each iteration of this loop assigns at least one job to its appropriate edge server. So, the computation complexity of this algorithm is $O(n^2mk)$, and SAFSA can calculate the pure Nash equilibrium of the introduced selfish allocation and fair scheduling game in a polynomial time. □

SAFSA runs in a central server with general knowledge of the system. This central server can be a broker in the edge layer or the cloud layer, and the output of this algorithm is the allocation and scheduling decision for the input jobs.

VIII. EXPERIMENTS

We simulate our joint allocation and scheduling scheme using an extended version of the CloudSim framework [13] and an initial implementation of SAFSA. The simulation environment consists of 50 IoT devices and 5 edge servers. We considered one computation resource of CPU, along with the link bandwidth, in each edge server. Edge servers in the simulations have link capacities from {1 GB/s, 5 GB/s, 10 GB/s}, and processing powers from {800 MIPS, 1500 MIPS, 2000 MIPS, 3300 MIPS}. Table 2 represents the parameters of the simulation environment.

We used Google cluster traces [57] representing resource requirements and usage data for tasks in several days of using Google machines. The traces provide information on job duration, resource demands, and resource usage. We used 500 traces in this analysis. Most resource requirements and usage in the traces include CPU, memory, disk space, and disk time fraction. However, the Google cluster traces do not have any information on bandwidth usage. Therefore,

Algorithm 1 Selfish Allocation and Fair Scheduling Algorithm (SAFSA)

Inputs: {T}: the set of input jobs; {E}: the set of edge servers; Ψ_{j_b}: the sum of weights of jobs assigned to the link bandwidth resource of every edge server R_j; Ψ_{j_c}: the sum of weights of jobs assigned to each computation resource c of every edge server R_j
for each job $i = 1 : n$ **do**
  $x_i = \arg\min_{j\in\{1:m\}}\{\max_{c=1:k}(\frac{T_{i_c}}{R_{j_c}}) + \frac{T_{i_{\kappa+1}}}{R_{j_{\kappa+1}}}\}$
end for
for each edge server $j = 1 : m$ **do**
  $\lambda_{j_{\kappa_{\kappa=1:k+1}}} = 0$ ▷ λ_{j_κ} is the cost of resource κ in R_j
end for
repeat
  for each edge server $j = 1 : m$ **do**
    Consider T_1 the job with $\max_{i:x_i=j}\left(\max_{c=1:k}(\frac{T_{i_c}}{R_{j_c}d_{i_c}}) + \frac{T_{i_{\kappa+1}}}{R_{j_{\kappa+1}}d_{i_{\kappa+1}}}\right)$ ▷ T_1 is the job with the highest priority in R_j
    if no job was assigned to R_j **then**
      $w_{1,j_{\kappa_{\kappa=1:k+1}}} = 1$
      $\Psi_{j_{\kappa_{\kappa=1:k+1}}} = w_{1,j_{\kappa_{\kappa=1:k+1}}}$
    else
      calculate w_{1,j_κ} according to equation (5)
      $\Psi_{j_{\kappa_{\kappa=1:k+1}}} += w_{1,j_{\kappa_{\kappa=1:k+1}}}$
    end if
    $\lambda_{j_{\kappa_{\kappa=1:k+1}}} += \frac{T_{1_{\kappa_{\kappa=1:k+1}}}}{R_{j_{\kappa_{\kappa=1:k+1}}} \times w_{1,j_{\kappa_{\kappa=1:k+1}}}} \times \Psi_{j_{\kappa_{\kappa=1:k+1}}}$
    $\{T\} = \{T\} - T_1$
  end for
  for each job i in $\{T\}$ **do**
    $x_i = \arg\min_{j\in\{1:m\}}\{\max_{c=1:k}(\lambda_{j_c} + \frac{T_{i_c}}{R_{j_c}w'_{i,j_c}}\Psi_{j_c}) + (\lambda_{j_{\kappa+1}} + \frac{T_{i_{\kappa+1}}}{R_{j_{\kappa+1}}w'_{i,j_{\kappa+1}}}\Psi_{j_{\kappa+1}})\}$ ▷ $w'_{i,j_{\kappa_{\kappa=1:k+1}}}$ is a temporary weight of T_i in R_j having $\Psi_{j_{c_{c=1:k}}}$, $\Psi_{j_{\kappa+1}}$
  end for
until $\{T\} = \emptyset$
return $X = \{x_1, x_2, \ldots, x_n\}$

we borrowed the instructions of [58] to model the bandwidth demand of jobs according to their CPU usage. Considering W indicates the number of CPU cycles used by an application, and L is the number of data bits as the input to the application, we have $W = L \times Y$, where Y is a random variable with Gamma distribution. We used the shape parameter $\alpha = 4$, and rate parameter $\beta = 200$, to generate Y, according to [58].

The experiments are performed on a personal computer with Intel(R) Core(TM) i5, 2.5 GHz CPU and 8 GB RAM. We divide the set of input jobs into n equally non-overlapping groups where group i indicates the input jobs of IoT device i. The input jobs are entered with a specific arrival rate. We used two performance metrics in the evaluations. The first is the average response time of jobs, and the second is the deadline miss rate, which is the number of jobs that take longer to complete than their deadlines divided by the total number of jobs. We have repeated all experiments at least 100 times, so each point on the plots is the average result of at least 100 different simulation runs. The proposed algorithm of SAFSA is compared with the following four methods:

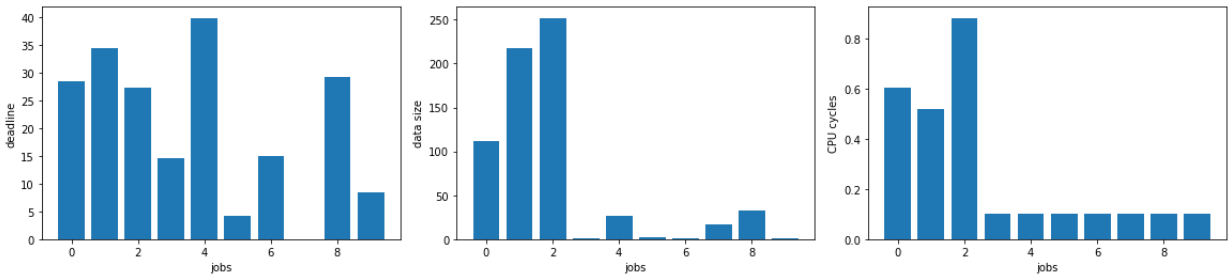
- Max-min+FCFS: In this scheme, the jobs are allocated to edge servers using the minimum dominated resource concept of DRF [6]. Also, the allocated jobs are scheduled in the edge servers' resources using the first come, first service (FCFS) algorithm [9]. itemMax-min+RR: This scheme also uses the DRF [6] method for allocating jobs. However, the allocated jobs are scheduled using the round-robin [30] algorithm.
- Greedy+FCFS: In this scheme, every job selects an edge server such that the sum of the ratio of resource demand to the available resource of the edge server for all resource types is minimized. Then the allocated jobs are scheduled using the FCFS algorithm.
- Greedy+RR: This scheme also allocates jobs greedily according to the sum of the ratio of resource demand to the available resource of the edge server for all resource types. This method schedules the jobs using the round-robin.

We called evaluated schemes that use the max-min allocation as max-min based schemes and the methods that use the greedy allocation as greedy based schemes.

A. EVALUATION EXPERIMENTS

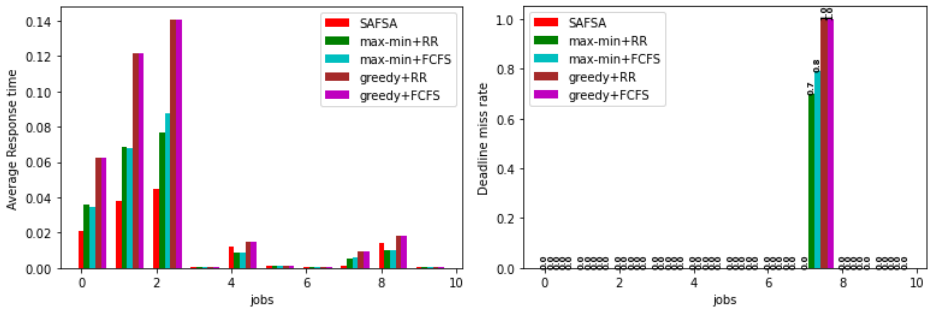
We first analyze response time and deadline miss rate for ten jobs of the dataset to study the ability of SAFSA compared to other evaluated schemes. The specifications of these jobs, including the deadline, the input data size, and the number of CPU cycles needed to complete the jobs, are represented in figure 2, respectively. Figure 3 shows the average results for 100 simulation runs of these jobs. The results confirm that response times for greedy based schemes with FCFS and RR scheduling are higher than all other methods. Except for two jobs of 4 and 8, where the response time of SAFSA is a little higher than max-min based schemes, SAFSA has a better response time for other jobs. As you can see in figure 2, deadlines for jobs 4 and 8 are higher than others. At the same time, their data size and required CPU cycles are not very high. So, the priority of these jobs in SAFSA is lower than other jobs having smaller deadlines and higher resource requests, which leads to a little higher response time than max-min based approaches. Nevertheless, thanks to their high deadlines, they can be completed within their deadlines with either SAFSA or max-min based approaches. In other words, SAFSA has 100% superior response time than greedy based schemes and 80% greater than that of max-min based schemes with RR and FCFS scheduling.

Regarding the deadline miss-rate of jobs, all compared schemes can execute all jobs except job 7 before their deadlines. It is due to the very small deadline of this job compared to other jobs. The greedy based schemes can not execute job 7 within its deadline in all 100 runs. Furthermore, max-min based methods missed this job's deadline 70 and 80 times,



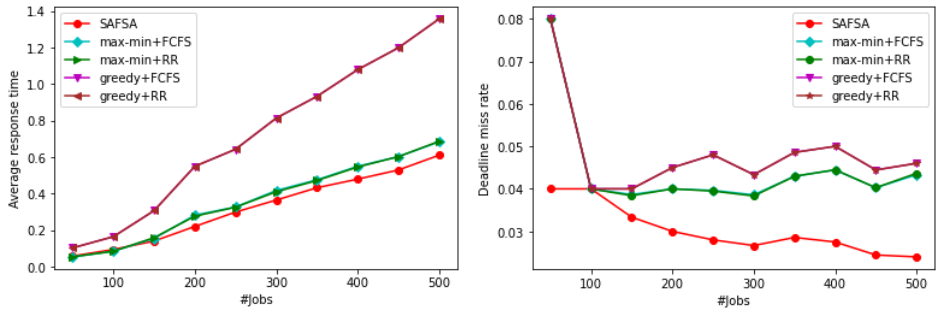
(a) deadline (b) data size (c) the number of CPU cycles

FIGURE 2. The specifications of 10 jobs from Google cluster traces.



(a) Response time (b) Deadline miss rate

FIGURE 3. Response time and deadline miss rate for 10 jobs in 100 simulation runs.



(a) Average response time (b) Deadline miss rate

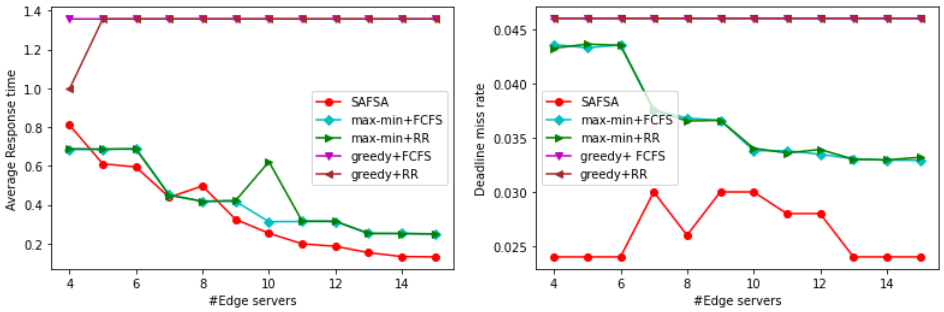
FIGURE 4. Average response time and deadline miss rate, in terms of the number of input jobs.

respectively. At the same time, SAFSA gives a high priority to job 7 and succeeded in executing all jobs within their deadlines. So, the deadline miss rate of our scheme is zero for all ten jobs.

1) IMPACT OF THE NUMBER OF INPUT JOBS

Figure 4 represents the performance of different schemes by increasing the number of input jobs to the system from 50 to 500. In this analysis, the arrival rate of jobs did not change. In all evaluated schemes, increasing the number of jobs increments the average response time. Greedy based schemes with RR and FCFS scheduling have a higher response time than other evaluated methods in all cases.

SAFSA attempts to simultaneously decrease the deadline miss rate and average response time by considering the shares of jobs allocated to each edge server. At the same time, approaches similar to max-min based only consider the resource request of the jobs. In these schemes minimizing the response time of jobs with high resource requests and deadlines can decrease the average response of the jobs. However, neglecting jobs with fewer requests and very short deadlines can cause them to miss deadlines. Therefore, max-min based approaches can compete with SAFSA in terms of the average response time, while their performance is not acceptable in minimizing the deadline miss rate of jobs. Since in SAFSA, the set of jobs assigned to each edge server affects the allocation decision of a new job, the performance of our method is robust by increasing the number of input jobs. This feature is proved in figure 4a, where the performance of SAFSA improves and distances from max-min based approaches by



(a) Average response time **(b)** Deadline miss rate

FIGURE 5. Average response time and deadline miss rate, in terms of the number of edge servers.

increasing the number of input jobs. Max-min based methods have equal response time for both RR and FCFS scheduling in all cases. It indicates the significant importance of allocating jobs to edge servers compared to scheduling in the servers.

On the other hand, in terms of deadline miss rate, SAFSA outperforms in comparison to all other schemes with significant performance. Increasing the number of input jobs increases the time needed to execute them over limited resources. However, in some cases, the jobs' deadline is large enough such that they can be executed at their deadlines. So, an appropriate allocation and scheduling approach can be robust against increasing the deadline miss rate as the number of jobs increments. Therefore there is not necessarily a reverse relation between the average response time and deadline miss rate of the jobs. This situation can be established in the analysis of this section and the following sections.

In this evaluation, by increasing the number of input jobs, the deadline miss rate increases in the evaluated schemes, while SAFSA prevents deadline violations when the number of input jobs increases. Results from this evaluation confirm that SAFSA is robust in minimizing average response time and deadline miss rate for the different number of input jobs.

2) IMPACT OF THE NUMBER OF EDGE SERVERS

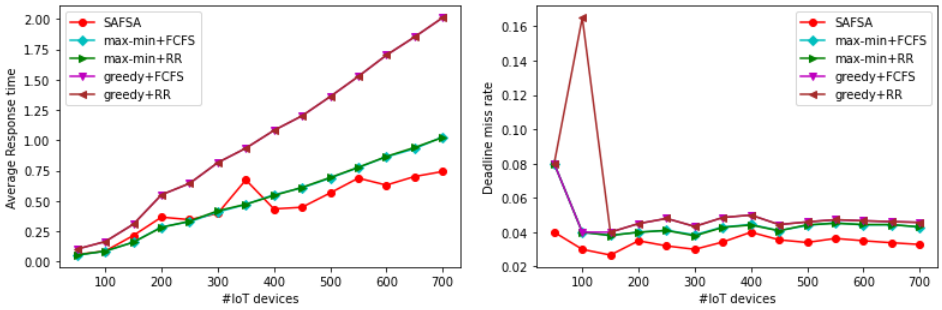
In this part, we have analyzed the performance of the proposed scheme in an edge computing system with 50 IoT devices entering 500 jobs when the number of edge servers changes from 4 to 15. Decreasing the number of edge servers leads to limitations in the communication and computation resources, which increases competition among jobs in allocating resources. A successful approach must complete more jobs within their deadlines when limited resources are available.

Figure 5 indicates greedy-based approaches' performance does not change by increasing the number of edge servers. This is because all jobs greedily select an edge server with higher resource capacity and do not assign to other edge servers, even if idle. It significantly reduces greedy allocation performance with RR and FCFS scheduling methods. However, the average response times of max-min based approaches and SAFSA decrease by increasing the number of edge servers. Since the jobs do not have many options to change their allocation strategy when there is a low number of edge servers, SAFSA's average response time is close to max-min based approaches in this case (figure 5). By incrementing the number of edge servers and increasing jobs' options for allocation strategies in SAFSA, its average response time difference with max-min based approaches increases. The configurations of edge servers are obtained arbitrarily according to table 2. So, the new edge server may not have better resource capacity than available edge servers, preventing jobs from choosing it for their initial allocation strategy. It can bring fluctuations in the results of SAFSA and max-min based approaches for the different number of edge servers.

According to figure 5b, the deadline miss rates of max-min based allocation schemes with RR and FCFS scheduling are decreasing by increasing the number of edge servers. However, they can not reach that of SAFSA, and it always succeeds in terms of deadline miss rate over other evaluated approaches, thanks to considering the jobs' deadlines besides their resource demands, in the introduced weighted time-sharing scheduling. This result confirms the ability of SAFSA to decrease both average response time and deadline miss rate, while max-min based approaches' success is just in terms of the average response time.

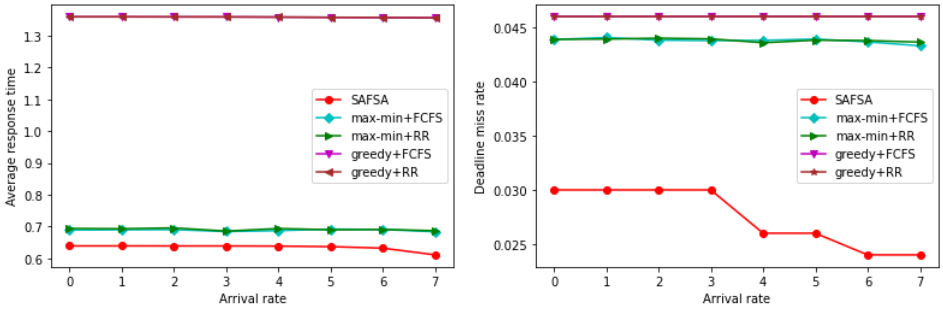
3) IMPACT OF THE NUMBER OF IoT DEVICES

Analyzing the methods by increasing the number of IoT devices from 50 to 700 is represented in Figure 6. Incrementing the number of IoT devices in the system leads to enlarging the input jobs, increasing the average response time of all evaluated schemes. However, the slopes of greedy based approaches' average response time curves are steeper than other schemes. It indicates the poor function of these methods for a large number of IoT devices. This performance is predictable from section VIII-A1 in analyzing the methods regarding the number of input jobs. Since both max-min based schemes and SAFSA pay attention to the resource requirements of the jobs, they provide comparable average response times (figure 6a). However, by increasing the



(a) Average response time (b) Deadline miss rate

FIGURE 6. Average response time and deadline miss rate in terms of the number of IoT devices.



(a) Average response time (b) Deadline miss rate

FIGURE 7. Average response time and deadline miss rate, in terms of jobs' arrival rate.

number of IoT devices, SAFSA's performance surpasses that of max-min based approaches. This result, along with that of section VIII-A1, confirms SAFSA's efficient and robust performance by increasing the number of input jobs and IoT devices.

On the other hand, figure 6b indicates the deadline miss rate of SAFSA is lower than that of other evaluated schemes for different numbers of IoT devices. It is because of considering the jobs' deadlines besides their resource demands in allocation and scheduling decisions of SAFSA. In contrast, other schemes like max-min based approaches only focus on jobs' resource demands. Although SAFSA has a similar average response time or even worse in some limited points in comparison to max-min based approaches, its outstanding ability to decrease the deadline miss rate of the jobs for all different numbers of IoT devices confirms SAFSA's capability in decreasing both response time and deadline miss rate of jobs by changing the number of IoT devices. This power is evident for a large number of IoT devices, a key feature of IoT applications.

4) IMPACT OF JOBS' ARRIVAL RATE

Figure 7 exhibits the performance of the analyzed methods with different arrival rates of 500 jobs to the edge computing system. We represent jobs' arrival rates with values from 0 to 7, where 0 indicates a very high job arrival rate and 7 indicates a very low arrival rate. By decreasing the arrival rate of jobs, the average response time diminishes in both SAFSA and max-min based schemes. Nevertheless, according to figure 7a, the average response time of SAFSA is lower than all other evaluated schemes for different arrival rates. Since SAFSA considers jobs' deadlines besides resource requirements on their allocation and scheduling, it decreases deadline miss rate by reducing jobs' arrival rates. In contrast, other methods do not significantly change their deadline miss rates for different arrival rates. This indicates the outstanding ability of our scheme in minimizing both response time and deadline miss rate for different arrival rates of jobs.

IX. CONCLUSION

We proposed a joint allocation and scheduling scheme for multi-user edge computing environments in which competition among users' jobs is modeled as a selfish game. We considered heterogeneous communication and computation resources in the edge computing system and the multi-resource requirements of IoT applications with deadline constraints. Also, we have focused on jobs with indivisible loads in which they are allocated and scheduled on one edge server. The satisfaction of jobs is considered with allocating resources proportional to their resource demands and deadlines. A selfish allocation game and a weighted time-sharing scheduling method were introduced to achieve this objective. By demonstrating the existence of pure Nash

equilibrium, we provided a greedy and polynomial time algorithm to achieve selfish allocation and fair scheduling solution called SAFSA. We assessed the performance of SAFSA using a real-world dataset of Google cluster [57], and an extended version of the CloudSim simulator [14]. The evaluations are performed in terms of average response time and the ratio of jobs with missed deadlines in an edge computing system with two types of resources (i.e., CPU and link bandwidth) for the edge servers. SAFSA's performance was compared with greedy and max-min allocations also FCFS and RR scheduling. SAFSA minimizes both measurements simultaneously, while others just focus on improving the average response time. The evaluations confirm this result for different circumstances of the edge computing environment, especially a large number of input jobs and IoT devices, a characteristic of IoT applications. To extend the current study, it is encouraging to consider the QoS requirements of users along with the constraints of resources such as energy. A further exciting topic for future research is the joint allocation and scheduling of dependent jobs.

ACKNOWLEDGMENT

The authors would like to thank Mohammad Amin Rayej for his help in providing the simulation environment.

REFERENCES

[1] R. Buyya and S. N. Srirama, *Fog and Edge Computing: Principles and Paradigms*. Hoboken, NJ, USA: Wiley, 2019.

[2] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the Internet of Things," *IEEE access*, vol. 6, pp. 6900–6919, 2017.

[3] H. Siar and M. Izadi, "Offloading coalition formation for scheduling scientific workflow ensembles in fog environments," *J. Grid Comput.*, vol. 19, no. 3, pp. 1–20, Sep. 2021.

[4] F. J. Miandashti, M. Izadi, A. A. N. Shirehjini, and S. Shirmohammadi, "An empirical approach to modeling user-system interaction conflicts in smart Homes," *IEEE Trans. Hum.-Mach. Syst.*, vol. 50, no. 6, pp. 573–583, Dec. 2020.

[5] M. Drozdowski, *Selected Problems of Scheduling Tasks in Multiprocessor Computer Systems*. Politechnika Poznańska, 1997.

[6] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types," in *Proc. NSDI*, vol. 11, 2011, p. 24.

[7] W. Wang, B. Li, and B. Liang, "Dominant resource fairness in cloud computing systems with heterogeneous servers," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2014, pp. 583–591.

[8] E. Meskar and B. Liang, "Fair multi-resource allocation in mobile edge computing with multiple access points," in *Proc. 21st Int. Symp. Theory, Algorithmic Found., Protocol Design Mobile Netw. Mobile Comput.*, Oct. 2020, pp. 11–20.

[9] J. L. Peterson and A. Silberschatz, *Operating System Concepts*. Boston, MA, USA: Addison-Wesley, 1985.

[10] X. Liu, Z. Qin, Y. Gao, and J. A. McCann, "Resource allocation in wireless powered IoT networks," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4935–4945, Jun. 2019.

[11] H. Zhang, Y. Zhang, Y. Gu, D. Niyato, and Z. Han, "A hierarchical game framework for resource management in fog computing," *IEEE Commun. Mag.*, vol. 55, no. 8, pp. 52–57, Aug. 2017.

[12] R. Tripathi, S. Vignesh, V. Tamarapalli, A. T. Chronopoulos, and H. Siar, "Non-cooperative power and latency aware load balancing in distributed data centers," *J. Parallel Distrib. Comput.*, vol. 107, pp. 76–86, Sep. 2017.

[13] R. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw., Pract. Exper.*, vol. 41, no. 1, pp. 23–50, 2011.

[14] M. A. Rayej, H. Siar, and M. Izadi, "WIDESim: A toolkit for simulating resource management techniques of scientific workflows in distributed environments with graph topology," 2022, *arXiv:2206.03538*.

[15] Q. Luo, C. Li, T. H. Luan, and W. Shi, "Minimizing the delay and cost of computation offloading for vehicular edge computing," *IEEE Trans. Services Comput.*, vol. 15, no. 5, pp. 2897–2909, Sep. 2022.

[16] S. Islam, S. Badsha, S. Sengupta, H. La, I. Khalil, and M. Atiquzzaman, "Blockchain-enabled intelligent vehicular edge computing," *IEEE Netw.*, vol. 35, no. 3, pp. 125–131, May 2021.

[17] M. D. Hossain, L. N. T. Huynh, T. Sultana, T. D. T. Nguyen, J. H. Park, C. S. Hong, and E.-N. Huh, "Collaborative task offloading for overloaded mobile edge computing in small-cell networks," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Jan. 2020, pp. 717–722.

[18] M. Kloock, P. Scheffe, I. Tulleners, J. Maczijewski, S. Kowalewski, and B. Alrifaee, "Vision-based real-time indoor positioning system for multiple vehicles," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 15446–15453, 2020.

[19] S. Malik, S. Ahmad, I. Ullah, D. H. Park, and D. Kim, "An adaptive emergency first intelligent scheduling algorithm for efficient task management and scheduling in hybrid of hard real-time and soft real-time embedded IoT systems," *Sustainability*, vol. 11, no. 8, p. 2192, Apr. 2019.

[20] T. M. Amert, "Enabling real-time certification of autonomous driving applications," Ph.D. dissertation, Dept. Comput. Sci., Univ. North Carolina Chapel Hill, Chapel Hill, NC, USA, 2021.

[21] J. Khamse-Ashari, I. Lambadaris, G. Kesidis, B. Urgaonkar, and Y. Zhao, "Per-server dominant-share fairness (PS-DSF): A multi-resource fair allocation mechanism for heterogeneous servers," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2017, pp. 1–7.

[22] M. Shafiee and J. Ghaderi, "On max-min fairness of completion times for multi-task job scheduling," in *Proc. IFIP Netw. Conf.*, 2020, pp. 100–108.

[23] E. Meskar and B. Liang, "MAGIKS: Fair multi-resource allocation game induced by Kalai-Smorodinsky bargaining solution," *IEEE Open J. Commun. Soc.*, vol. 3, pp. 797–810, 2022.

[24] P. Poullie, T. Bocek, and B. Stiller, "A survey of the state-of-the-art in fair multi-resource allocations for data centers," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 1, pp. 169–183, Mar. 2018.

[25] M. Mukherjee, M. Guo, J. Lloret, R. Iqbal, and Q. Zhang, "Deadline-aware fair scheduling for offloaded tasks in fog computing with inter-fog dependency," *IEEE Commun. Lett.*, vol. 24, no. 2, pp. 307–311, Feb. 2020.

[26] M. Zhao, W. Li, L. Bao, J. Luo, Z. He, and D. Liu, "Fairness-aware task scheduling and resource allocation in UAV-enabled mobile edge computing networks," *IEEE Trans. Green Commun. Netw.*, vol. 5, no. 4, pp. 2174–2187, Dec. 2021.

[27] A. Kishor, R. Niyogi, and B. Veeravalli, "Fairness-aware mechanism for load balancing in distributed systems," *IEEE Trans. Services Comput.*, vol. 15, no. 4, pp. 2275–2288, Jul. 2022.

[28] D. Grosu and A. T. Chronopoulos, "Noncooperative load balancing in distributed systems," *J. Parallel Distrib. Comput.*, vol. 65, no. 9, pp. 1022–1034, 2005.

[29] G. Xing, X. Xu, H. Xiang, S. Xue, S. Ji, and J. Yang, "Fair energy-efficient virtual machine scheduling for Internet of Things applications in cloud environment," *Int. J. Distrib. Sensor Netw.*, vol. 13, no. 2, Feb. 2017, Art. no. 155014771769489.

[30] R. V. Rasmussen and M. A. Trick, "Round Robin scheduling—A survey," *Eur. J. Oper. Res.*, vol. 188, no. 3, pp. 617–636, 2008.

[31] H. Yuan, G. Tang, X. Li, D. Guo, L. Luo, and X. Luo, "Online dispatching and fair scheduling of edge computing tasks: A learning-based approach," *IEEE Internet Things J.*, vol. 8, no. 19, pp. 14985–14998, Oct. 2021.

[32] A. J. Neha, "An improved round Robin CPU scheduling algorithm," *Iconic Res. Eng. J.*, vol. 1, no. 9, pp. 82–86, 2018.

[33] S. Zouaoui, L. Boussaid, and A. Mtibaa, "Priority based round Robin (PBRR) CPU scheduling algorithm," *Int. J. Electr. Comput. Eng. (IJECE)*, vol. 9, no. 1, p. 190, Feb. 2019.

[34] M. M. Tajwar, M. N. Pathan, L. Hussaini, and A. Abubakar, "CPU scheduling with a round Robin algorithm based on an effective time slice," *J. Inf. Process. Syst.*, vol. 13, no. 4, pp. 941–950, 2017.

[35] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. K. Baruah, J. E. Gehrke, and C. G. Plaxton, "A proportional share resource allocation algorithm for real-time, time-shared systems," in *Proc. 17th IEEE Real-Time Syst. Symp.*, Dec. 1996, pp. 288–299.

[36] S. Ramabhadran and J. Pasquale, "Stratified round Robin: A low complexity packet scheduler with bandwidth fairness and bounded delay," in *Proc. Conf. Appl., Technol., Archit., Protocols Comput. Commun. (SIGCOMM)*, 2003, pp. 239–250.

[37] M. Mukherjee, V. Kumar, D. Maity, R. Matam, C. X. Mavromoustakis, Q. Zhang, and G. Mastorakis, "Delay-sensitive and priority-aware task offloading for edge computing-assisted healthcare services," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2020, pp. 1–5.
[38] J. Meng, H. Tan, X.-Y. Li, Z. Han, and B. Li, "Online deadline-aware task dispatching and scheduling in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1270–1286, Jun. 2020.
[39] S. Lee, S. Lee, and S.-S. Lee, "Deadline-aware task scheduling for IoT applications in collaborative edge computing," *IEEE Wireless Commun. Lett.*, vol. 10, no. 10, pp. 2175–2179, Oct. 2021.
[40] L. Liu, Q. Fan, and R. Buyya, "A deadline-constrained multi-objective task scheduling algorithm in mobile cloud environments," *IEEE Access*, vol. 6, pp. 52982–52996, 2018.
[41] V. Cardellini, V. D. N. Persone, V. D. Valerio, F. Facchinei, V. Grassi, F. L. Presti, and V. Piccialli, "A game-theoretic approach to computation offloading in mobile cloud computing," *Math. Program.*, vol. 157, no. 2, pp. 421–449, 2016.
[42] Y. Li, W. Dai, X. Gan, H. Jin, L. Fu, H. Ma, and X. Wang, "Cooperative service placement and scheduling in edge clouds: A deadline-driven approach," *IEEE Trans. Mobile Comput.*, vol. 21, no. 10, pp. 3519–3535, Oct. 2022.
[43] X. Lyu, H. Tian, W. Ni, Y. Zhang, P. Zhang, and R. P. Liu, "Energy-efficient admission of delay-sensitive tasks for mobile edge computing," *IEEE Trans. Commun.*, vol. 66, no. 6, pp. 2603–2616, Jun. 2018.
[44] A. Naouri, H. Wu, N. A. Nouri, S. Dhelim, and H. Ning, "A novel framework for mobile-edge computing by optimizing task offloading," *IEEE Internet Things J.*, vol. 8, no. 16, pp. 13065–13076, Aug. 2021.
[45] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 379–423, 1948.
[46] C. E. Shannon, "A mathematical theory of communication," *ACM SIGMOBILE Mobile Comput. Commun. Rev.*, vol. 5, no. 1, pp. 3–55, 2001.
[47] J. Du, L. Zhao, J. Feng, and X. Chu, "Computation offloading and resource allocation in mixed fog/cloud computing systems with min-max fairness guarantee," *IEEE Trans. Commun.*, vol. 66, no. 4, pp. 1594–1608, Apr. 2017.
[48] Y. Mao, J. Zhang, Z. Chen, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3590–3605, Dec. 2016.
[49] A. Iyer, C. Rosenberg, and A. Karnik, "What is the right model for wireless channel interference?" in *Proc. 3rd Int. Conf. Quality Service Heterogeneous Wired/Wireless Netw.*, 2006, pp. 1–10.
[50] M. Noura and R. Nordin, "A survey on interference management for device-to-device (D2D) communication and its challenges in 5G networks," *J. Netw. Comput. Appl.*, vol. 71, pp. 130–150, Aug. 2016.
[51] C. D. Nwankwo, L. Zhang, A. Quddus, M. A. Imran, and R. Tafazolli, "A survey of self-interference management techniques for single frequency full duplex systems," *IEEE Access*, vol. 6, pp. 30242–30268, 2018.
[52] A. Padmanabhan and A. Tolli, "Interference management via user clustering in two-stage precoder design," in *Proc. IEEE 19th Int. Workshop Signal Process. Adv. Wireless Commun. (SPAWC)*, Jun. 2018, pp. 1–5.
[53] M. J. Osborne, *An Introduction to Game Theory*, vol. 3. New York, NY, USA : Oxford Univ. Press, 2004.
[54] S. Tadelis, *Game Theory: An Introduction*. Princeton, NY, USA: Princeton Univ. Press, 2013.
[55] T. Roughgarden, "Algorithmic game theory," *Commun. ACM*, vol. 53, no. 7, pp. 78–86, 2010.
[56] D. Fotakis, S. Kontogiannis, E. Koutsoupias, M. Mavronicolas, and P. Spirakis, "The structure and complexity of Nash equilibria for a selfish routing game," in *Proc. Int. Colloq. Automata, Lang., Program.* Berlin, Germany: Springer, 2002, pp. 123–134.
[57] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: Format + schema," Google, Mountain View, CA, USA, White Paper, 2011, pp. 1–14.
[58] W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, and D. O. Wu, "Energy-optimal mobile cloud computing under stochastic wireless channel," *IEEE Trans. Wireless Commun.*, vol. 12, no. 9, pp. 4569–4581, Sep. 2013.

HAJAR SIAR is currently pursuing the Ph.D. degree with the Distributed and Multi-Agent Systems Laboratory, Department of Computer Engineering, Sharif University of Technology, Tehran, Iran. Her research interests include cloud/edge computing, distributed and multi-agent systems, game theory, and machine learning.

MOHAMMAD IZADI received the B.Sc., M.Sc., and Ph.D. degrees in computer engineering, the M.Sc. degree in philosophy of science from the Sharif University of Technology, Tehran, Iran, and the Ph.D. degree in computer science from Leiden University, The Netherlands. He is currently an Associate Professor and the Head of the Distributed and Multi-Agent Systems Laboratory, Department of Computer Engineering, Sharif University of Technology. His research interests include distributed and multi-agent systems, logic in computer science, semantics, game theory, and theory of computation.