



Task scheduling in edge-fog-cloud architecture: a multi-objective load balancing approach using reinforcement learning algorithm

Fatemeh Ramezani Shahidani¹ · Arezoo Ghasemi¹ · Abolfazl Toroghi Haghghat¹  · Amin Keshavarzi²

Received: 24 September 2022 / Accepted: 26 December 2022 / Published online: 5 January 2023
© The Author(s), under exclusive licence to Springer-Verlag GmbH Austria, part of Springer Nature 2023

Abstract

The rapid development of internet of things (IoT) gadgets and the increase in the rate of sending requests from these devices to cloud data centers resulted in congestion and consequently service provisioning delays in the cloud data centers. Accordingly, fog computing emerged as a new computing model to address this challenge. In fogging, services are provisioned at the edge of the network using devices with computing and storage capabilities, which are located through the way to connect IoT devices to cloud data centers. Fog computing aims to alleviate the computing load in data centers and cut the delay of requests down, notably real-time and delay-sensitive requests. To achieve these goals, vitally important challenges such as scheduling requests, balancing the load, and reducing energy consumption, which affects performance and reliability in the edge-fog-cloud computing architecture, should be considered into account. In this paper, a reinforcement learning fog scheduling algorithm is proposed to address these challenges. The experimental results indicate that the proposed algorithm raises the load balance and diminishes the response time compared to the existing scheduling algorithms. Additionally, the proposed algorithm outperforms other approaches in terms of the number of used devices.

✉ Abolfazl Toroghi Haghghat
haghghat@qiau.ac.ir

Fatemeh Ramezani Shahidani
Fatemehramezani84@yahoo.com

Arezoo Ghasemi
arezoo.qasemi214@gmail.com

Amin Keshavarzi
keshavarzi@miau.ac.ir

¹ Faculty of Computer and Information Technology Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran

² Department of Computer Engineering, Marvdasht Branch, Islamic Azad University, Marvdasht, Iran

Keywords Scheduling · Edge computing · Fog computing · Cloud computing · Load balancing · Reinforcement learning

1 Introduction

Pervasive use of Internet of Things (IoT) has led to generation of a huge volume of data in the IoT networks [1]. In such networks, a wide range of heterogeneous devices communicate through network facilities. It is anticipated that approximately 40 billion gadget will be interconnected through the Internet by 2025 [2]. Due to this, traditional computing models such as cloud computing fails to meet IoT systems requirements such as Quality of Service (QoS) guarantee [3]. Fog computing as a complement computing paradigm for cloud computing aims to address this challenge, and provision services in the edge of network and near to the source of data [4, 5].

Fog networking was coined by Cisco to tackle the problems of sending IoT devices requests to the cloud computing [6]. They defined fog computing as a architecture allocating resources near the source of data. Fogging as the intermediate between cloud computing and the edge computing, provides the facilities for end users to process their data and reduce the latency for end the customers and workload for the cloud data centers [2, 7].

Fogging provides a powerful computing facility that enables processing at the edge as well as interacting with the cloud. Recently, fogging achieves breakthroughs in health care, industry, agriculture, smart cities, environmental, transportation, and climate change control. Fog nodes characteristics such as heterogeneity and distribution of resources and their limitations in processing, storage and memory capacities make task scheduling a challenging problem [2, 8].

Scheduling in cloud-fog framework has rarely received considerable attention. The main goal of task scheduling algorithms is to diminish task running time and enhance performance. Cloud providers should utilize efficient scheduling methods to address the requirements of their customers and raise resource efficiency, which can lead to the development of green information technology. Tasks scheduling algorithms have been proposed with different goals such as reducing energy consumption, resource wastage, cost, delay, response time, but one of the vitally important goals is the load balancing. Load balancing distributes requests or tasks of a computing environment among resources with the aim of increasing performance and reliability. Based on our knowledge a few researchers have paid attention to the task scheduling with the goal of balancing the load in fog networking [2].

In this paper, a reinforcement learning based task scheduling algorithm with the aim of balancing the load, diminishing the average of response time, and reducing energy consumption is proposed which is called RLFS (Reinforcement Learning Fog Scheduling). In the proposed algorithm, the learning agent has a table of action qualities (Q-Table) and at the end of each learning episode, the values of this quality of action table are updated according to the degree of desirability of the performed action. In each episode, the agent-learner uses the value table of his actions and the action selection policy defined for it and based on the state of the environment, selects an action from among the actions allowed in that state and executes it on the environment. The learning

agent performs the action, and after that the state changes. Then, the environment returns a feedback signal (reward signal) and a new state of the environment to the learning agent according to the degree of desirability of the performed action (the degree of desirability is determined according to the learning goal). The learning agent updates the value table of its actions according to the feedback signal received from the environment. This work makes the learning agent able to choose the action that has the most feedback in the future episodes [9].

This paper is structured as follows. A brief overview of related works is given in section. In the third section the proposed architecture is presented. We propose the proposed algorithm in Sect. 4. In the fifth part, the proposed algorithm is evaluated and we explain the experiments settings and compare the results. Our conclusions are drawn in the sixth section, and the areas for further research are determined in final section.

2 Related work

In recent years, a large and growing body of algorithms have been proposed to address scheduling problem in fog computing. Table 1 demonstrates a comparison of related works in terms of pros. and cons., and an analytical overview of related works and proposed approach have been provided in Table 2. As these tables depict, a few researchers have paid attention to the task scheduling with the aim of balancing the load in fog networking.

In [10], a task scheduling algorithm called IEGA was proposed to diminish the response time in the fog environment. In this research, authors applied an improved genetic algorithm to raise the quality of provided services. The IEGA method comprises two general steps. In the first step, Mutation rate and Cross over are modified to search more of the problem space (possible combinations of permutations). In the second step, some of solutions are mutated based on a defined probability to avoid getting stuck in the local optimum. The proposed algorithm was compared with other algorithms in terms of carbon emissions, energy consumption, and performance of the fitness function. The simulation demonstrates that the proposed method outperform other methods in all criteria.

Fellir et al. [11] proposed a multi-agent task scheduling algorithm in fog computing. They consider task priority, waiting time, and resource status that serve the most important tasks in their proposed method. Regarding to this, in the scheduling process, they first update the task priority value and then allocate resources based on this priority (priority of tasks, waiting time, status). Also, the priority of the tasks is updated based on the priorities of the dependent tasks that exist. In the proposed model, high priority tasks are processed first, until a higher priority task is queued. Regarding the results of the ifogsim simulator, the proposed method demonstrates good performance.

Binh et al. [12] proposed a task scheduling algorithm in the fog computing environment called TCaS which aims to make a balance between execution time and operational costs. In addition, the proposed algorithm can be flexibly adapted to various users with different demand where some want to prioritize the execution time and others want to complete their tasks with limited resources. The results of simulation

Table 1 Comparing proposed approach and related work in terms of pros. and cons

Year	Cons.	Pros.	Method	References
2021	<ul style="list-style-type: none"> • Fail to consider the cloud environment • Lack of considering resource efficiency • Run time longer than EGA algorithm 	Improved power consumption and improved response time	Genetics	[10]
2020	<ul style="list-style-type: none"> • Unlikely to consider resource efficiency • Lack of taking bandwidth into account 	Reduce energy consumption and cost		[11]
2018	<ul style="list-style-type: none"> • Total cost more than BLA algorithm • Less likely to consider energy consumption 	Better achieve QoS and makespan	Genetics	[12]
2019	<ul style="list-style-type: none"> • Not considering energy consumption • Resource efficiency • Bandwidth 	Achieve low latency Multi-source fit across tasks	Deep Reinforcement Learning	[13]
2021	<ul style="list-style-type: none"> • Bandwidth is not considered 	Along with optimizing the response time and energy consumption, it ensures the reliable execution of tasks	Atamata learner	[14]

Table 1 continued

Year	Cons.	Pros.	Method	References
	<ul style="list-style-type: none"> ● Fail to consider resource efficiency and cloud environment ● Not considering energy consumption ● Fail to consider resource efficiency and cloud environment ● Not considering the bandwidth, r 	Reduced service latency—improved overall performance stability	Genetics	[15]
2020	<ul style="list-style-type: none"> ● Resource efficiency and cloud environment 	Reducing energy consumption		[16]
2018	<ul style="list-style-type: none"> ● Not considering the bandwidth and resource efficiency 	Improving the device performance, energy consumption optimizing, and balancing th load among robots in factories Reduction of transmission delay and computational costs	Improved particle swarming	[17]
2021	<ul style="list-style-type: none"> ● Limited to use in factories ● Not considering energy consumption in decision making 	Better performance than other algorithms by combining metrics,Performance, security and cost in scheduling decisions	AHP , TOPSIS	[18]
2019	<ul style="list-style-type: none"> ● Not considering the cloud environment ● Not considering energy utilization, resource efficiency, and cloud environment 	Reduce makespan execution time, Transfer time and in terms of criteria, Determining the distance has performed better	Moth-flame	[19]
2021	<ul style="list-style-type: none"> ● Not considering energy utilization, resource efficiency, and bandwidth 	Reduce the number of missed tasks	Genetics	[20]
2019	<ul style="list-style-type: none"> ● The execution time is longer than the distance based algorithm 	Increase user satisfaction Reducing energy consumption	Fuzzy Systems	[21]

Table 1 continued

Year	Cons.	Pros.	Method	References
2021	<ul style="list-style-type: none"> Lack of scalability 	Reduce makespan, Increasing the success rate in performing tasks, Reduce latency reduce execution time	Fuzzy Systems	[22]
2020	<ul style="list-style-type: none"> The amount of energy consumption and carbon emission is higher than the BALS algorithm 	Reduce makespan, Reduce costs	Shahin-Harris	[23]
2021	<ul style="list-style-type: none"> Poor resource selection decisions when select resources from other fog nodes 	Reduce the number of resources used, Resource efficiency, load balancing		[4]
2020	<ul style="list-style-type: none"> Not considering energy utilization, resource utilization, and load balancing 	Reduce total latency	Genetic algorithm	[24]
2018	<ul style="list-style-type: none"> Not considering latency and response time 	Increase resource utilization and strike a balance between loads	DRAM	[25]
2018	<ul style="list-style-type: none"> Not considering energy consumption and load balancing 	Increasing security and decreasing response time	PSO and SA	[26]
2022	<ul style="list-style-type: none"> Not considering energy consumption and resource utilization 	Decreasing failure rate and service time	DRL	[27]
2021	<ul style="list-style-type: none"> Not considering load balancing and resource utilization 	Decreasing energy consumption	Clustering	[28]
2021	<ul style="list-style-type: none"> Not considering resource utilization, and load balancing 	Reduce energy consumption and makespan	Stepwise Frequency Scaling	[29]
2022	<ul style="list-style-type: none"> Little increase in execution time 	Creating load balance among resources, Reducing the number of active resources, Reducing energy consumption	Reinforcement learning	Proposed approach

Table 2 Comparing related works and proposed method regarding considered parameters in scheduling

Ref	Bandwidth	Load balancing	QoS	makespan	security	Response	Resource	Cost	Run time	Delay	Reliability	Energy
[10]						✓						✓
[11]								✓	✓			✓
[12]			✓	✓								
[13]							✓		✓	✓		
[14]						✓					✓	✓
[15]										✓		
[16]												✓
[17]	✓											✓
[18]					✓			✓				
[19]				✓					✓			
[20]						✓						
[21]			✓									
[22]				✓		✓			✓	✓		✓
[23]				✓				✓				
[24]						✓				✓		
[25]		✓					✓					✓
[26]					✓			✓				
[27]		✓				✓				✓		
[28]												✓
[29]				✓								✓
[4]	✓	✓					✓					✓
Proposed approach	✓	✓				✓						✓

depict that the this method achieves better QoS than other methods, and outperforms others in striking a balance between efficiency and cost.

In [13], authors put forward an online task scheduling approach in the fog computing environment which was called FairTS. They applied Deep-Reinforcement Learning algorithm for online task scheduling. The concept of online task scheduling refers to making resource assignment decisions for tasks that are continuously generated from end users. This is a challenging problem due to the various uncertainties that appear in fog computing systems. Their proposed approach can learn directly from experience to shorten the task slowness while make sure multi-resource fairness.

Ghanavati et al. [14] offered a task scheduling algorithm using the Learning automaton technique, called DFTLA. The aim of this algorithm is to tolerate errors and optimize QoS and energy consumption. It is vitally important to provide a task scheduling approach, which guarantees the reliability of tasks execution while optimizing the response time in the fog computing environment. Therefore, they proposed a new DFTLA dynamic fault tolerant task scheduling method. DFTLA specifies an efficient allocation of tasks to fog nodes using an automaton that is capable of making backup decisions for high-priority tasks. Using this method, tasks are run on fog processor nodes based on an efficient estimate of runtime. This approach models the task execution time using two main criteria including static execution time and variable execution time in the form of normal probability distribution. To optimize consumption of energy, they introduced an energy-aware decision-making method that is used by the proposed task scheduling approach. This method enables the scheduling algorithm to consider the remaining energy of the fog node before assigning a new task to other fog nodes. This algorithm guarantees the reliable execution of the tasks, and optimize energy consumption and response time.

Sun et al. [15] applied NSGAI improved genetic algorithm to schedule resources in fog computing environment. First, they proposed a new fog computing architecture comprises three layers, terminal, core and edge. The composition and performance of fogging accurately are described by this architecture. Then, a two-level scheduling model is proposed including two components, inter resource scheduling and intra resource scheduling. Finally, to deal with multi-objective optimization problems, a new resource scheduling approach has been proposed, which effectively cuts the service delay. The MATLAB simulation approves that the proposed method reduces the delay and enhances the stability of execution of tasks.

In [16], an energy-aware scheduling algorithm in fog networking environment called PTOM was proposed by authors. They investigated an integer linear programming approach and to find the optimal solution they proposed a critical path (CP) based method. They divide the problem into several states and show that the problem can be solved in multinomial time complexity. The results demonstrates that although this method has a large workload, it can perform better than the existing methods and also consumes less energy.

Wan et al. [17] put forward an energy-aware scheduling and load balancing algorithm in smart factory. They proposed an intelligent scheduling and load balancing method called ELBS. In order to balance the load of the production cluster, the workload of the fog device is modeled as an energy consumption schema. To determine the priority of tasks on the production cluster side, they use the improved particle

swarm optimization (PSO) method. Finally, to schedule production clusters in a distributed manner, they use a multi-agent framework. The results show that the proposed scheduling and load balancing method provides optimal task for robots.

Subbaraj et al. [18] tried to address the problem of scheduling using decision-making methods such as AHP and TOPSIS. Their goal was to provide a resource-based model of tasks in a fog computing environment. They considered some performance criteria for resource task mapping such as MIPS, RAM, Storage, Up link, Down link, Uplink bandwidth, Downlink bandwidth, Reliability, MIPS cost, Memory cost, Storage cost and Bandwidth cost into account. To evaluate the performance of fog devices, they utilized two different multi-attribute decision-making methods. In the first method, they used the AHP hierarchical analysis process to calculate the weight and ranking of fog devices. In the second method, AHP was used to calculate the weight, and based on the weights obtained by AHP, the TOPSIS algorithm was implemented in order to rank fog machines. Then, fog devices were assigned to tasks based on their rank. The simulation results depict that their technique has superior performance over other scheduling algorithms in the fog environment by combining performance, security, and cost criteria in scheduling decisions.

The Moth-flame optimization algorithm is another approach which was argued for task scheduling algorithm [19]. Researchers aim to minimize the execution time of tasks and satisfy the quality of service requirements of Cyber-Physical System applications in this research. Minimization of task execution and transfer time are considered as target functions in the proposed algorithm. The weights of the criteria that should be minimized in the scheduling process were considered equal in the algorithms presented in the past. To approve the degree of importance of the criteria in the scheduling, the corresponding weight coefficients should be considered for them. They used weighting coefficients to indicate the degree of importance in the scheduling. Allocation of weights to the criteria provides the possibility of finding a better optimal solution. They simulated their proposed algorithm using the iFogsim simulator and compared it with the state of the art in terms of execution time, transfer time and distance determination criteria.

To minimize deadline constraints in a hybrid fog-cloud computing environment, Aburukba et al. [20] developed a genetic algorithm-based optimization algorithm for scheduling computing requests from fixed IoT devices. Depending on the nature of the request, these requests have different deadlines. The fog layer receives requests from IoT devices and assigns them to the fog-cloud according to the scheduling model in an attempt to minimize communication delays. In this method, they take into account delays such as transmission delays, queuing delays in different network devices, and task execution delays in cloud and fog layers. The proposed model is customized with the aim of minimizing the missed deadline. A well-designed crossover operator and a convenient representation of a solution (chromosome) are provided in this research. Also, a penalization procedure is included in the genetic algorithm in order to penalize possible solutions that are not compliant with the constraints. This makes impossible solutions to generate new chromosomes less likely to be chosen.

Benblidia et al. [21] proposed a task scheduling approach using fuzzy system in fog computing environment. Performing tasks in the fog computing environment requires an efficient scheduling mechanism that takes into account preferences of user

orders and fog cloud requirements. They utilized two parameters called the highest satisfactory proportion (GSP) and the lowest satisfactory proportion (LSP) to detect similarities. According to experimental findings, the proposed approach balances average user satisfaction with execution delay and energy consumption while satisfying user preferences.

In [22], a task scheduling approach called FLRTS was proposed by authors. They used the fuzzy logic method to split the tasks between cloud and fog. It exploits the task requirements such as storage, computation, and bandwidth, and their constraints such as size of data and deadline to select the appropriate node to execute the tasks submitted in the fog layer with heterogeneous resources. This algorithm has been simulated with the iFogsim simulator and compared with other algorithms in terms of task success, lifetime, average circulation time and delay.

Abdel-Basset et al. [23] proposed a task scheduling algorithm in the fog environment, using Shahin Harris meta-heuristic algorithm called HHOLS. Their proposed algorithm is an energy-aware approach based on Shaheen Harris algorithm with the aim of improving the quality of service provided to users. Mutation improves the quality of solutions due to its ability to balance the workload between all VMs. For further improvement, a local search approach is integrated with HOLS. They simulated their proposed algorithm with Java programming language and compared it with numerous meta-heuristic methods in terms of cost, energy consumption, flow time, and carbon flow rate.

In [4], a task scheduling algorithm called LBSSA was proposed with the aim of reliability in the cloud-fog architecture. First, using a classifier module, they classified the tasks into three categories: real-time, important, and non-real time tasks. Then, based on the type of task that entered the fog node, a decision is made for scheduling. Based on this, when a task is real-time or important, it is tried to use the resources inside the fog node for allocation in the first step. If the fog node does not have the necessary resources for that task, other fog nodes are used. Also, to reduce the workload, non-real time tasks are entrusted to the cloud.

Aburukba et al. [24] modeled request scheduling problem in fog computing using integer programming and applied Genetic Algorithm (GA) to address this problem with the aim of cutting total delay of requests down. They customized GA to minimize the delay in a reasonable time, and compared their work with three different approaches. The simulation demonstrated that the proposed method diminish the total latency compared the related work.

Yasmeen et al. [25] proposed a provisioning algorithm for fogging. This method considered security of resource communication and aimed to reduce the response time. They utilized PSO algorithm and combined it with Simulated Annealing (SA) to avert getting stuck in local optima and rapid convergence. Their approach outperformed the related work in simulation experiments.

A Dynamic Resource Allocation Method for fog computing environment, called DRAM, was proposed by Xu et al. [26]. This algorithm aimed to make a balance between resources in fog layer. In this regard, resources was classified based on type of resources and start time of services. Also, during resource allocation process, if the utilization of selected resource was less than predefined threshold, the task was migrated to another resource. Authors compared their approach to the related work in

terms of the variance of load balancing, the mean of resource utilization, and the number of used resources. The simulation results depicted that this approach outperform the related work.

In [27], a task scheduling approach was proposed using deep reinforcement learning (DRL) for the Edge environment. The authors applied DRL to the task scheduling problem with the aim of declining failure rate and service time. The proposed approach can learn from previous action and can find the optimum solution in the absence of a concrete mathematical model of the environment. Also, they utilized Deep-Q-Network(DQN) to address the curse of dimensionality issue of task scheduling. The simulation results of EdgeCloudSim simulator demonstrated that the proposed approach depicted the better performance in terms of service time, failure rate and VM utilization in comparison of the state of the art.

A study by Hao et al. [28] examined applying clustering algorithm in task scheduling problem in the edge computing environment. They supposed that each node reaches green energy using some devices and the energy of each node can transmit using Energy Internet (EI) technology. The authors clustered nodes with the aim of declining intra cluster distance. The contribution of this research is the definition of distance which was defined as a combination of green energy cost during transmitting files and the time of transmitting files.

In [29] a scheduling approach was proposed to diminish energy consumption and makespan. The proposed algorithm schedule delay-sensitive tasks with low computational need in fog layer and the tasks with high computational need and less sensitivity to the delay in cloud layer. Also, they utilized Stepwise Frequency Scaling approach, which is a deadline-aware approach, to reduce energy consumption. The simulation results depicts that this method plunge energy consumption by 50 percent but fails to affect makespan.

3 Edge-cloud-fog architecture

Figure 1 depicts edge-cloud-fog architecture. This architecture comprises three layers: Edge, fog, and cloud. Each device of the edge layer is bind to a fog node and sends the request to that fog node. Also, each cloud node provides resources in the response of requests sent to it from the edge layer. Each fog node contains a manager node that receives requests from the edge layer and put them in the queue. Requests are entered to the classifier module in order, and the type of requests are determined. The requests types include Real Time, Non-real time, and Important. Real time requests are very sensitive to delay and have deadlines. Important requests are sensitive to delay, but there is no deadline for them. Non-real time requests are hardly sensitive to delay.

When the type of request was determined, the request is entered into the scheduler module for scheduling. Indeed, scheduler provisions appropriate resource to respond to the request. Due to limitation of available resources in the fog layer, it fails to meet the resources requirements of all requests. So, non-real time requests are forwarded to the cloud layer so that there exist enough resources in the fog layer to provision for the remaining requests which are sensitive to delay. By this, it is tried to avert sending real time and important requests to the cloud layer as mush as possible. Also, if the fog

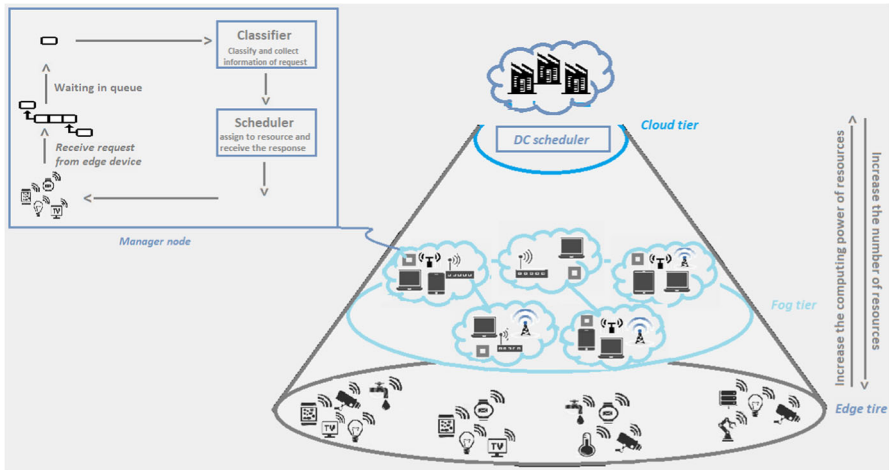


Fig. 1 Edge-cloud-fog architecture

node fails to provide enough resources to respond to a request, it will seek help from the resources available in other fog nodes. The request is sent to the cloud If other fog nodes refuse to provide resources for that request [4, 30].

4 Reinforcement learning fog scheduling (RLFS) algorithm

As it was stated in the previous section, requests of IoT devices are classified into three categories: Real time, Important, and Non-real time. In the RLFS algorithm, all non-real time requests are sent to the cloud because they are rarely sensitive to time, but other categories, Real time and Important, should be answered in the fog layer due to the limitation of response time. To scheduling Real time and Important requests, Reinforcement learning method is applied. Reinforcement learning technique is utilized in RLFS algorithm in two levels. In the first, all Real time and Important requests are tried to be answered in the fog nodes in which the request has been received. A level of reinforcement learning is used to select an appropriate resource to send the request to that resource, and if the fog node that received the request fails to provide the resources to run that request, it borrows the resources from other fog nodes. Selecting a fog node to borrow its resources is carried out using another level of reinforcement learning. The process of reinforcement learning and the details of the RLFS algorithm are described in the following.

4.1 The first level of reinforcement learning (provisioning resources for scheduling the received request at the fog node)

At this level of learning, the received requests of Real time and Important types in the fog node are scheduled. At the first, it is tried to select appropriate resources to

meet the requirements of request inside the fog node. In the following, the process of selecting appropriate resources inside the fog node is described.

- **Environment modes** Similarly to the types of requests received in the fog node, there are two general modes for the environment, Real time and Important.
- **Authorized actions** Each learning agent includes a series of actions in each state of the environment, which in each learning episode chooses one of those actions based on the defined action selection policy and performs it on the environment. In this algorithm, in each state of the environment, the allowed actions include all the resources that are available in the fog node and have the ability to meet the requirements of request. If the fog node receiving the request does not have any resources available to respond to the request, the agent enters the second level of reinforcement learning to select an appropriate fog node to send the request.
- **Action Selection Policy** Here we utilize the ϵ -greedy policy to select an action in each state of the environment. Based on this policy, the action that has the highest action quality in the current state in the quality of action table (Q-Table) is selected with a very high probability (95%) and other actions with a probability of ϵ (5%) as actions in that episode. The value of an action is higher if its value is less than the rest of the actions since this is a minimization problem.
- **Rewards** After the action is selected using the action selection policy and applied to the environment, the amount of reward (feedback signal) of the performed action should be returned to the agent. In this weighted sum algorithm, the amount of internal load imbalance of the selected resource, the amount of load imbalance between the resources from the processor side and the resource waste from the processor and memory side are calculated and returned to the agent as a reward signal. Resource wastage is calculated by Eqs. 1, load imbalance among resources is calculated by 2, and internal load imbalance is calculated by Eq. 3.

$$R_i^w = \frac{\sum_{k=1}^r (|R_i^k - \min(R_i^k)|) + \epsilon}{\sum_{k=1}^r u_i^k} \tag{1}$$

where $R_i^k = 1 - u_i^k$ is the remaining resource k in the i-th device. $\min(R_i^k)$ It represents the lowest amount of remaining resources (among processor and memory resources) in the i-th device. Also, u_i^k indicates the efficiency of resource k in the i-th device. Normalized values are used in this equation.

$$unevenness(cpu) = \sqrt{\frac{1}{M} \cdot \sum_{j=1}^M \left(\frac{L_j^{cpu}}{\bar{L}^{cpu}} - 1 \right)^2} \tag{2}$$

where M depicts the number of resources, L_j^{cpu} represents the processor load in the j-th source, and (\bar{L}^{cpu}) represents the average processor load among all resources. It is important to balance the amount of remaining resources in each unique resource and in each dimension, which is referred to as internal load imbalance. Figure 2 shows two scheduling schemes of six requests on two resources 1 and 2.

The cube marked as T_i represents the amount of processor memory allocated to the i -th request. The orange cube shows the remaining amount of processor and memory. Figure 2a shows a scheduling scheme. It can be seen that there are more memory resources in resource 1, but the remaining processor is not enough. Hence, resource 1 cannot answer every request instance that requires a normal amount of CPU and memory. Unlike resource 1, resource 2 has a lot of remaining CPU, but not enough memory. Thus, resource 2 also cannot answer every request that requires a normal amount of memory and processor. In another scheme shown in Fig. 2b, the remaining capacity of both resources 1 and 2 is more balanced than in Fig. 2a. This balance helps a resource to respond to more requests simultaneously. This leads to answer most of the requests inside the fog node and to send requests to other fog nodes or the cloud less often. We used Eq. 3 to measure internal load imbalance in multiple resource usage [9].

$$disequilibrium(j) = \sqrt{\frac{1}{R} \cdot \sum_{r=1}^R \left(\frac{L_j^r - \bar{L}_j}{\bar{L}_j} \right)^2} \quad (3)$$

Here, $R = 2$ is considered as the number of dimensions, where we consider the two dimensions of processor and memory. \bar{L}_j and L_j^r demonstrate the average load of all resources and resource load of r respectively. Therefore, the reward is calculated according to the Eq. 4.

$$Reward_R = \left(\frac{1}{a} \cdot unevenness(cpu) \right) + \left(\frac{1}{a} \cdot disequilibrium(j) \right) + \left(\frac{1}{a} \cdot R_i^w \right) \quad (4)$$

where $a=3$ and equal to the number of parameters.

- **Update the value table of actions (Q-Table)** After receiving the reward signal from the environment, the learning agent updates the quality of action table in each episode. In this algorithm, after receiving the reward signal, the agent should update the quality of action table related to this level of learning, which we call QR. For this purpose, we utilize Eq. 5 to update the QR table.

$$QR_{s,a} = (1 - \alpha)QR_{s,a} + \alpha[R + \gamma QR_{s',a'}] \quad (5)$$

4.2 The second level of reinforcement learning (selecting appropriate fog node to send a request to another fog node)

As we stated, when the fog node receives a request from IoT devices and does not have the appropriate resources to respond to that request, it uses other fog nodes to respond to that request. To select the appropriate fog node to send the request to that fog node, the manager node utilizes the second level of reinforcement learning.

- **Environment modes** At this level, the environment modes are the same as the types of requests, i.e., two modes, Real time and Important.

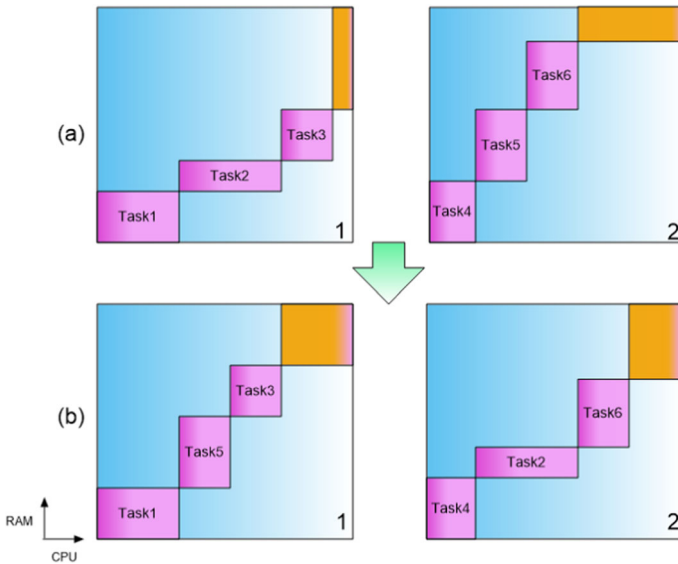


Fig. 2 Two scheduling schemes. **a** Without considering internal load balancing. **b** Considering internal load balancing

- **Authorized actions** At this level of learning, the virtual actions of agent include fog nodes that have available resources to respond to the desired request.
- **Action Selection Policy** To select an action among the allowed actions in each state of the environment, we apply the ϵ -greedy policy.
- **Rewards** We calculate the available bandwidth as bonus signal $Reward_F$, because the request transmission time between fog nodes is vitally important, and the determining parameter in this measure is the amount of link bandwidth,
- **Update the value table** As before, after the agent receives the reward signal, the quality of action table should be updated. Here, the quality of action table corresponding to the second level of reinforcement learning called QF is updated using Eq. 6.

$$QF_{s,a} = (1 - \alpha)QF_{s,a} + \alpha[R + \gamma QF_{s',a'}] \quad (6)$$

Algorithm 1 shows the pseudo code of RLFS algorithm. In the pseudo-code, the input and output parameters are provided in the first to sixth lines. Then, in the 8th and 9th lines, the value of the actions in both levels of reinforcement learning are initialized with zero. In the 10th to 36th lines, the process of checking requests and scheduling is carried out inside a loop for each request. Based on this, if the request type is non-real time, the 11th line is run first, then the request is sent to the cloud in the 12th line. Otherwise, the process of selecting the resource for the request should be conducted. In the 14th line, the type of request is specified first, and the type of request determines the state of the environment.

Then, in the 15th line, we get a list of all the resources in the fog node that can respond to the request using the AvailableR function. Then in the 16th line, if the retrieved list is not empty, there is a resource to assign the request, the list is

investigated, and in the 17th line, an action (a resource from the list of resources) is selected using the ChoiceAction function and the $\epsilon - greedy$ action selection policy. Then, in the 18th line, the desired request is sent to the selected resource for processing. Then, in the 19th line, the reward for the performed action is calculated, this reward includes the weighted sum of the internal load imbalance regarding the processor and memory, the amount of load imbalance among the resources inside the fog node regarding the processor, and the resource wastage.

After calculating the reward in the 20th and 21st lines, two variables s' , a' are quantified to update the value table. Then, in the 22nd line, based on equation 5, the value table is updated. Then, in the 23th line, the state of the environment is updated. If the fog node receiving the request does not have available resources to assign, we enter the second level of reinforcement learning. Regarding this, in the 25th line, a list containing all the fog nodes that have available resources to respond to the desired request is constituted. Then it is controlled in the 26 line, if this list is empty, the request is sent to the cloud layer in the 27th line. Otherwise, a fog node is selected based on the $\epsilon - greedy$ action selection policy in the 29 line. In line 30, the request is sent to the selected node. Then, in the 31 line, according to the first level of reinforcement learning, a resource in the new fog node is selected to send a request to that resource. In the 32nd line, the reward for the performed action is calculated that this reward is the amount of available bandwidth. Finally, in line 33 and 34, s' , a' is set to update the value of the table.

In the 35th line, the quality of action table is updated using equation 6, and finally, in the 36th line, the environment state is updated. Also, if there is a need to transfer the request to another fog node or cloud level, and the waiting time for the release of the resource at the same level is less than the time of transferring the request to another level, that request will not be transferred to another level and the request waits at the same level until the required resource is released.

5 Experimental environment and results

The proposed approach is compared with the LBSSA algorithm, which was reviewed in the related works section and is one of the most recent algorithms in the field of task scheduling in the fog networking. In the experiments, the proposed algorithm was evaluated regarding various criteria such as the amount of load balancing, the average of response time, the number of used devices, the average efficiency and the execution time. To implement these algorithms, we ran CloudSim simulator on a laptop with 4 GB of RAM, a maximum frequency of 2.13 GHz, and a 64-bit Windows 11 operating system.

5.1 Simulation setup

The network environment was a three-layer network consisting of cloud, fog, and edge layers. The cloud layer had three data centers in each data center ten virtual machines were running. Also, the fog layer contained a hundred nodes (Fog nodes), each node

Algorithm 1: RLFS Algorithm

```

1 Input :
2   RequestList    \\List of received requests from IoT devices
3   ResourceList   \\List of available resources
4   FogList        \\List of fog nodes
5 Output :
6   Resource       \\Selected Resource
7 Initialization :
8   QR = 0;        \\Action value table for resource
9   QF = 0;        \\Action value table for fog nodes
10 for inti = 0; i < RequestList.size(); i ++ do
11   if The request is non-RealTime then
12     | The request is transferred to the cloud
13   else
14     State = Classifire(Request)    \\ Classification type of request
15     AvaList = AvailableR(ResourceList) \\ resource in fog that can service
16     if AvaList.size! = 0 then
17       a = ChoiceAction(State, AvailableList); \\ Selects resource ( $\epsilon$ -greedy)
18       Assigne request to resource
19       Reward =  $1/\alpha(RW(a)) + (1/\alpha)(inter - LB(a)) + (1/\alpha)(intra - LB(a))$ ;
20       S' =Next request type ;
21       a' = ChoiceAction(S', ResourceList);
22       QRs,a =  $(1 - \alpha)Q_{s,a} + \alpha[Reward + \gamma Q_{s',a'}]$ ;
23       S = S';
24   if AvaList.size == 0 then
25     FList = AvailableF(FogList)
26     if FList.size==0 then
27       | The request is transferred to the cloud
28     else
29       a = ChoiceAction(State, FList); \\ Selects fog node ( $\epsilon$ -greedy)
30       Assigne request to resource of fog node(a)
31       Select the source using the first level of reinforcement learning
32       Reward = AvailableBW;
33       S' =Next request type ;
34       a' = ChoiceAction(S', ResourceList);
35       QFs,a =  $(1 - \alpha)Q_{s,a} + \alpha[Reward + \gamma Q_{s',a'}]$ ;
36       S = S';

```

contained a number of resources to respond to the requests of the IoT devices. The fog nodes were connected via a network with one gigabyte bandwidth. Also, fog nodes were connected with the cloud layer via communication links with a bandwidth of ten gigabytes.

5.2 Data set

In the experiments, LCG (Large Hadron Collider Computing Grid) dataset¹ was used to simulate the requests of the IoT devices. The simulation results were analyzed in

¹ Available at https://www.cs.huji.ac.il/labs/parallel/workload/l_lcg/.

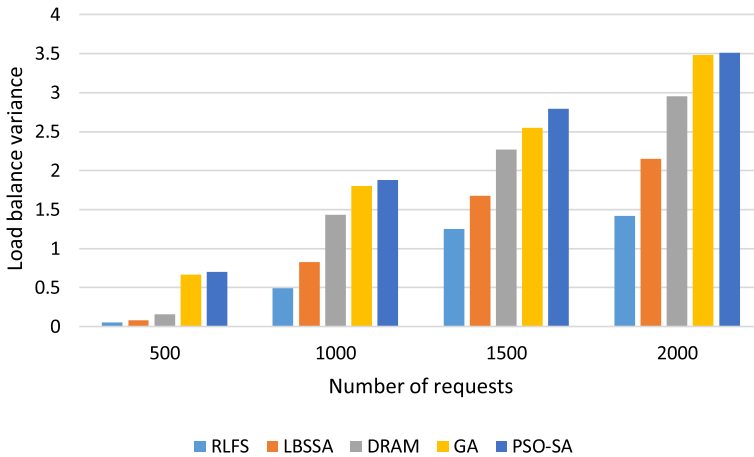


Fig. 3 Load balance variance for different number of requests (500, 1000, 1500, 2000)

four different scenarios consisting of 500, 1000, 1500 and 2000 requests from the IoT devices.

5.3 Comparison and review of results

We provide the results of comparing the proposed algorithm (RLFS) with LBSSA, DRAM, GA, and PSO-SA algorithms in terms of load balance, the number of used devices, the average of response time, and execution time in this section.

5.3.1 Load balance

Figure 3 demonstrates the load balance variance comparison of proposed RLFS algorithm with LBSSA, DRAM, GA, and PSO-SA algorithms in the case of designed scenarios with various requests from IoT devices. According to this chart, even though algorithms which considers load balancing as an objective, including the proposed RLFS algorithm, LBSSA and DRAM, demonstrates better load balance in all four scenarios, our proposed algorithm outperforms others. In comparison to the LBSSA algorithm, the RLFS algorithm enhances the load balance by 42%, 41%, 36% and 34% for the first to fourth scenarios respectively, since the load balancing parameter was considered as one of the parameters in calculating the amount of reward for the action performed by the agent. In the LBSSA algorithm for allocating tasks to fog devices, in the first, all devices are sorted regarding the failure rate, and then the first task is assigned to the first device that has the ability to respond to it. As a result, task scheduling is merely carried out based on the resource failure rate. Also, in the DRAM method, tasks are classified based on the type of resource and start time of service. In this approach, if the performance of selected resource be less than a threshold during the resource allocation process, task is migrated to the another resource.

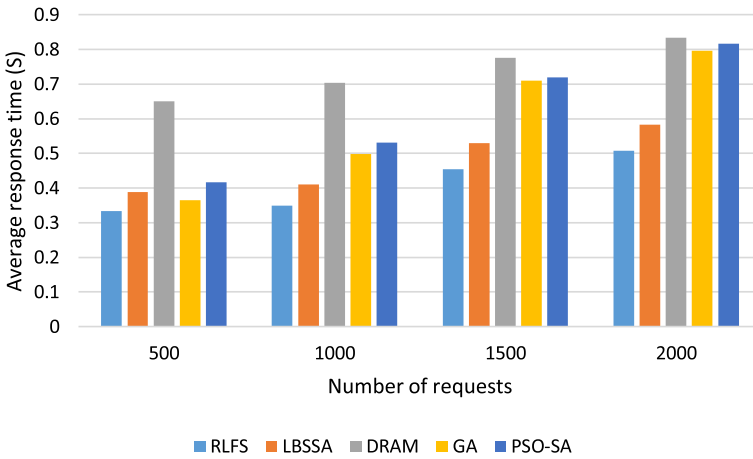


Fig. 4 Average response time for different number of requests (500, 1000, 1500, 2000)

In contrast, in the proposed method, the agent selects a device for task scheduling according to the value of each action in the Q table and regarding the ϵ -greedy action selection policy. The value table of actions is updated at the end of each episode based on the amount of reward received. The amount of reward consists of three parameters: the amount of internal load imbalance of the device selected to respond to the task, the amount of load imbalance among devices and the amount of resource wastage in terms of processor and memory. Calculating the amount of load imbalance among fog resources leads to outperforming the other algorithms in terms of the load balance parameter.

5.3.2 Average response time

Figure 4 shows the average response time for different scenarios. As it has been depicted in the figure, the proposed algorithm outperform other algorithms in all scenarios. In comparison with LBSSA, our proposed method was able to achieve better results by 14%, 15%, 14%, and 13% than the LBSSA algorithm in all scenarios. As discussed in the previous chapter, considering the amount of internal load imbalance in the reward calculation makes nodes able to respond to more tasks. This causes more tasks to be answered within fog nodes and fewer tasks to be transferred to other fog or cloud nodes for processing, and consequently reduces the response time.

Also, if there is a need to transfer requests to other fog nodes, considering that in the second level of reinforcement learning, the bandwidth of the fog node is taken into consideration and the request is transferred to a fog node which has more bandwidth. This scales down the time of transferring the request to another node happens, and consequently reduces the response time. So, by considering the amount of internal load imbalance in the first level of reinforcement learning and considering the amount of bandwidth in the second level of reinforcement learning, the average response time in the proposed algorithm is diminished in comparison with other algorithms.

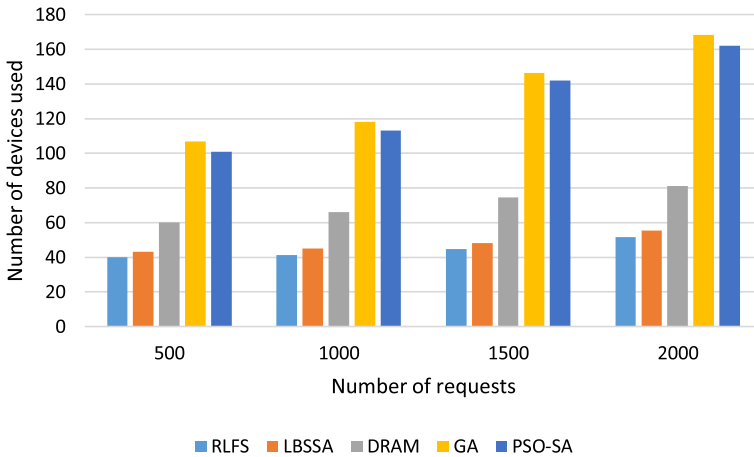


Fig. 5 Number of used devices for different number of requests (500, 1000, 1500, 2000)

In addition, by forwarding non-real time requests to the cloud layer in RLFS and LBSSA algorithms, more resources are available for important and real time requests that leads to declining average response time in these algorithms. From the other side, in GA, decreasing delay is considered as the single objective. Due to this, GA outperforms DRAM and PSO-SA in terms of average response time.

5.3.3 Number of used devices

Figure 5 depicts the number of used devices in different scenarios. As shown in the figure, the proposed algorithm was able to use fewer devices and after that the LBSSA achieve the second rank. Thus, in different scenarios, the proposed algorithm has been able to use less number of devices by 7 to 9 percent compared to the LBSSA algorithm. As discussed in the previous sections, one of the parameters considered in calculating the reward of the action performed by the agent is the amount of internal load imbalance. The amount of internal load imbalance means that a resource is balanced in all aspects of the processor and memory resources.

For example, there is no case with 95% efficiency in terms of processor but 10% efficiency in terms of memory. When this state occurs, even though the resource is 90% free in terms of memory, it cannot be used due to the high efficiency of the processor. Considering this, it becomes possible to assign more tasks to one resource. This makes the proposed algorithm able to use fewer devices when it tries schedule the same number of tasks in comparison with the LBSSA algorithm. Thanks to the continuous considering of scheduling tasks on appropriate resources LBSSA achieves better results compared with other algorithms. Also, DRAM attains the third rank since it consider migrating tasks to other nodes when the performance is less than the threshold.

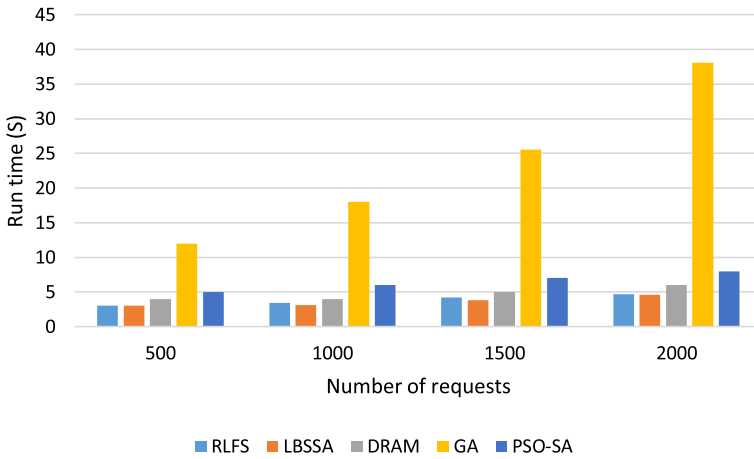


Fig. 6 Execution time for different number of requests (500, 1000, 1500, 2000)

5.3.4 Execution time

The results of execution time of all algorithms are summarized in Fig. 6. From this figure we can observe that the LBSSA has minimum run time and the GA the maximum one. Also, the proposed algorithm has more execution time than the LBSSA algorithm except the first scenario which the execution time of both algorithms is the same. In terms of execution time, the proposed algorithm has performed weaker than the LBSSA algorithm. The proposed algorithm has more processing than the LBSSA algorithm due to the update of the value of action table, the selection of the action based on the Q table, and the implementation of the reinforcement learning algorithm in two levels. Indeed, in the other parameters such as load balance and the number of used devices, the proposed algorithm outperforms the LBSSA algorithm, and the slight increase in execution time can be neglected.

From the other side, GA has the maximum run time since it is a population base algorithm. Indeed, it generates a set of solutions and evaluates them to find the best, and these operations are time consuming.

6 Conclusion and future works

The growing popularity of IoT has led to sharp rise of smart devices that request cloud services. Transferring this large volume of data to the cloud caused network congestion and increased latency. Fogging addresses mentioned problems by providing fast answers for real time requests, but due to limitation of resources, there is a need for effective scheduling of requests for these resources, which guarantees their efficient utilization. This paper has designed an approach for task scheduling in the edge-fog-cloud architecture using reinforcement learning. We implemented our proposed method using Cloudsim simulator and compared it with LBSSA, DRAM, GA, and PSO-SA algorithms in terms of load balance, number of used devices, resource

efficiency and execution time. The simulation results have demonstrated that in the edge-fog-cloud setting, the proposed method has outperformed other algorithms in terms of load balance and average response time. Also, in terms of the number of used devices, it has presented better results. However, in terms of execution time, the LBSSA algorithm has had less execution time than the proposed algorithm, but due to the significant improvement of the proposed algorithm in terms of number of devices and load balance, the small increase in execution time can be ignored. This is because our proposed algorithm considers the amount of load imbalance in calculating the reward, and results in responding more tasks by fog nodes. Also, updating action table, selection the appropriate action, and overload of implementing RL in two levels leads to weaker performance of our algorithm in term of execution time.

Utilizing the fuzzy logic to calculate the reward is proposed as the future work. All three criteria of internal load imbalance, load imbalance among devices and resource efficiency can be considered as three inputs of the fuzzy system and the fuzzy system output can be returned as a reward to the learning agent. Also, synchronization between cloud and fog layers can be considered into account in the future studies. Mobility of IoT devices is another issue which can be taken into consideration as the future work.

Funding The authors declare that no funds, grants, or other support were received during the preparation of this manuscript.

Declarations

Conflict of interest The authors have no relevant financial or non-financial conflict of interests to disclose.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. Doryanizadeh V, Keshavarzi A, Derikvand T, Bohlouli M (2021) Energy efficient cluster head selection in internet of things using minimum spanning tree (eemst). *Appl Artif Intell* 35(15):1777–1802
2. Sarrafzade N, Entezari-Maleki R, Sousa L (2022) A genetic-based approach for service placement in fog computing. *J Supercomput* 78(8):10854–10875
3. Keshavarzi A, Haghighat AT, Bohlouli M (2021) Clustering of large scale qos time series data in federated clouds using improved variable chromosome length genetic algorithm (cqga). *Expert Syst Appl* 164:113840
4. Alqahtani F, Amoon M, Nasr AA (2021) Reliable scheduling and load balancing for requests in cloud-fog computing. *Peer-to-Peer Netw Appl* 14(4):1905–1916
5. Madhura R, Elizabeth BL, Uthariaraj VR (2021) An improved list-based task scheduling algorithm for fog computing environment. *Computing* 103(7):1353–1389
6. Azizi S, Shojafar M, Abawajy J, Buyya R (2022) Deadline-aware and energy-efficient iot task scheduling in fog computing systems: a semi-greedy approach. *J Netw Comput Appl* 201:103333
7. Khan T, Tian W, Zhou G, Ilager S, Gong M, Buyya R (2022) Machine learning (ml)-centric resource management in cloud computing: a review and future directions. *J Netw Comput Appl* 66:103405
8. Hosseinioun P, Kheirabadi M, Tabbakh SRK, Ghaemi R (2020) A new energy-aware tasks scheduling approach in fog computing using hybrid meta-heuristic algorithm. *J Parallel Distrib Comput* 143:88–96
9. Ghasemi A, Toroghi Haghighat A (2020) A multi-objective load balancing algorithm for virtual machine placement in cloud data centers based on machine learning. *Computing* 102(9):2049–2072
10. Abdel-Basset M, Mohamed R, Chakraborty RK, Ryan MJ (2021) Iega: an improved elitism-based genetic algorithm for task scheduling problem in fog computing. *Int J Intell Syst* 36(9):4592–4631

11. Fellir F, El Attar A, Nafil K, Chung L (2020) A multi-agent based model for task scheduling in cloud-fog computing platform. In: 2020 IEEE international conference on informatics, IoT, and enabling technologies (ICIoT). IEEE, pp 377–382
12. Binh HTT, Anh TT, Son DB, Duc PA, Nguyen BM (2018) An evolutionary algorithm for solving task scheduling problem in cloud-fog computing environment. In: Proceedings of the ninth international symposium on information and communication technology, pp 397–404
13. Bian S, Huang X, Shao Z (2019) Online task scheduling for fog computing with multi-resource fairness. In: 2019 IEEE 90th vehicular technology conference (VTC2019-Fall). IEEE, pp 1–5
14. Ghanavati S, Abawajy J, Izadi D (2020) Automata-based dynamic fault tolerant task scheduling approach in fog computing. *IEEE Trans Emerg Top Comput* 6:66
15. Sun Y, Lin F, Xu H (2018) Multi-objective optimization of resource scheduling in fog computing using an improved nsga-ii. *Wirel Pers Commun* 102(2):1369–1385
16. Tan H, Chen W, Qin L, Zhu J, Huang H (2020) Energy-aware and deadline-constrained task scheduling in fog computing systems. In: 2020 15th International conference on computer science & education (ICCSSE). IEEE, pp 663–668
17. Wan J, Chen B, Wang S, Xia M, Li D, Liu C (2018) Fog computing for energy-aware load balancing and scheduling in smart factory. *IEEE Trans Ind Inform* 14(10):4548–4556
18. Subbaraj S, Thiyagarajan R (2021) Performance oriented task-resource mapping and scheduling in fog computing environment. *Cognit Syst Res* 70:40–50
19. Ghobaei-Arani M, Souri A, Safara F, Norouzi M (2020) An efficient task scheduling approach using moth-flame optimization algorithm for cyber-physical system applications in fog computing. *Trans Emerg Telecommun Technol* 31(2):e3770
20. Aburukba RO, Landolsi T, Omer D (2021) A heuristic scheduling approach for fog-cloud computing environment with stationary iot devices. *J Netw Comput Appl* 180:102994
21. Benblidia MA, Brik B, Merghem-Boulahia L, Esseghir M (2019) Ranking fog nodes for tasks scheduling in fog-cloud environments: A fuzzy logic approach. In: 2019 15th international wireless communications & mobile computing conference (IWCMC). IEEE, pp 1451–1457
22. Ali HS, Rout RR, Parimi P, Das SK (2021) Real-time task scheduling in fog-cloud computing framework for iot applications: a fuzzy logic based approach. In: 2021 International Conference on Communication Systems & NETworkS (COMSNETS). IEEE, pp 556–564
23. Abdel-Basset M, El-Shahat D, Elhoseny M, Song H (2020) Energy-aware metaheuristic algorithm for industrial-internet-of-things task scheduling problems in fog computing applications. *IEEE Internet Things J* 8(16):12638–12649
24. Aburukba RO, AliKarrar M, Landolsi T, El-Fakih K (2020) Scheduling internet of things requests to minimize latency in hybrid fog-cloud computing. *Future Gener Comput Syst* 111:539–551
25. Yasmeen A, Javaid N, Rehman OU, Iftikhar H, Malik MF, Muhammad FJ (2018) Efficient resource provisioning for smart buildings utilizing fog and cloud based environment. In: 2018 14th International wireless communications & mobile computing conference (IWCMC). IEEE, pp 811–816
26. Xu X, Fu S, Cai Q, Tian W, Liu W, Dou W, Sun X, Liu AX (2018) Dynamic resource allocation for load balancing in fog environment. *Wirel Commun Mob Comput* 6:66
27. Zheng T, Wan J, Zhang J, Jiang C (2022) Deep reinforcement learning-based workload scheduling for edge computing. *J Cloud Comput* 11(1):1–13
28. Hao Y, Cao J, Wang Q, Du J (2021) Energy-aware scheduling in edge computing with a clustering method. *Future Gener Comput Syst* 117:259–272
29. Ijaz S, Munir EU, Ahmad SG, Rafique MM, Rana OF (2021) Energy-makespan optimization of workflow scheduling in fog-cloud computing. *Computing* 103(9):2033–2059
30. Guevara JC, da Fonseca NL (2021) Task scheduling in cloud-fog computing systems. *Peer-to-Peer Netw Appl* 14(2):962–977

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.