Journal of Network and Systems Management (2022) 30:54
https://doi.org/10.1007/s10922-022-09664-6

Real-Time Task Scheduling Algorithm for IoT-Based Applications in the Cloud–Fog Environment

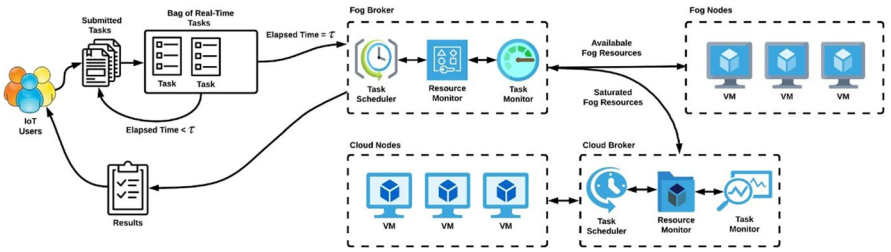
A. S. Abohamama[1] · Amir El-Ghamry[1] · Eslam Hamouda[1,2]

Received: 11 July 2021 / Revised: 22 April 2022 / Accepted: 27 May 2022 /
Published online: 2 July 2022

Abstract
IoT applications have become a pillar for enhancing the quality of life. However, the increasing amount of data generated by IoT devices places pressure on the resources of traditional cloud data centers. This prevents cloud data centers from fulfilling the requirements of IoT applications, particularly delay-sensitive applications. Fog computing is a relatively recent computing paradigm that extends cloud resources to the edge of the network. However, task scheduling in this computing paradigm is still a challenge. In this study, a semidynamic real-time task scheduling algorithm is proposed for bag-of-tasks applications in the cloud–fog environment. The proposed scheduling algorithm formulates task scheduling as a permutation-based optimization problem. A modified version of the genetic algorithm is used to provide different permutations for arrived tasks at each scheduling round. Then, the tasks are assigned, in the order defined by the best permutation, to a virtual machine, which has sufficient resources and achieves the minimum expected execution time. A conducted optimality study reveals that the proposed algorithm has a comparative performance with respect to the optimal solution. Additionally, the proposed algorithm is compared with first fit, best fit, the genetic algorithm, and the bees life algorithm in terms of makespan, total execution time, failure rate, average delay time, and elapsed run time. The experimental results show the superiority of the proposed algorithm over the other algorithms. Moreover, the proposed algorithm achieves a good balance between the makespan and the total execution cost and minimizes the task failure rate compared to the other algorithms.

Extended author information available on the last page of the article

Graphical Abstract



Keywords Scheduling optimization · Fog computing · Cloud–fog system · IoT applications · Bag-of-tasks applications

1 Introduction

Recently, Internet of Things (IoT) technology has played a significant role in different aspects of everyday life. The IoT and its relevant technologies have extended Internet connectivity to include a myriad of devices (e.g., sensors, machines, vehicles, and wearable devices) in addition to traditional smart devices [1]. These devices provide various services, such as health monitoring, intelligent traffic control, smart retail and logistics, and vehicular networking. The data generated by these devices must be processed to extract necessary information for IoT applications. However, the processing and storage capabilities of IoT devices cannot process this amount of data [2]. On the other hand, cloud computing is a distributed computing model that provides an on-demand pool of resources over the Internet [3]. The inherent limitations of IoT devices (e.g., short battery life, low computing power, and limited storage) can be addressed by assigning tasks that consume high resources to more powerful cloud nodes [4].

According to the Information Handling Services (IHS) Markit, the number of Internet-connected IoT devices will grow from 30.7 billion in 2020 to 75.4 billion in 2025, which will result in unprecedented amounts of generated data. Cloud datacenters are not able to fulfill the requirements of IoT applications [4]. Additionally, network congestion and transmission delay are inevitable and degrade the quality of service (QoS) for delay-sensitive/deadline-constrained IoT applications such as healthcare, smart cities, smart transportation, and online gaming [5]. These problems have motivated experts to extend cloud resources to the edge of the network.

Fog computing was first proposed by Cisco to reduce the burden of cloud datacenters by extending cloud computing to the edge of the network [6]. Fog computing consists of a variety of devices, including controllers, cellular base stations, embedded servers, access points, gateways, switches, routers, and surveillance cameras [7]. It offers many benefits, such as conserving the network bandwidth, reducing the energy consumption, supporting the mobility and geographical distribution of IoT devices, and achieving location awareness and low latency for delay-sensitive IoT

applications [6, 8]. However, the computing and storage capabilities of fog nodes are insufficient for large IoT applications such as big data analytics. Hence, cooperation between fog nodes and cloud nodes is necessary. This integration introduced a new computing model called cloud–fog computing [9].

Cloud–fog computing can support delay-sensitive and computing-intensive IoT tasks by jointly exploiting cloud and fog resources. The three-tier architecture (see Fig. 1) is the widely adopted architecture of cloud–fog computing. IoT devices tier consists of GPS-equipped IoT devices (e.g., sensors, smart phones, wearable devices, and smart vehicles) that enable end users to submit their tasks. The fog computing tier comprises edge devices such as routers, switches, and gateways. It provides computing and storage facilities close to data generation. Cloud computing tier consists of a large pool of resources in the form of datacenters [6, 10].

A cloud–fog system is a highly distributed computing paradigm that consists of heterogeneous hosts with a wide range of computing resources. Hence,

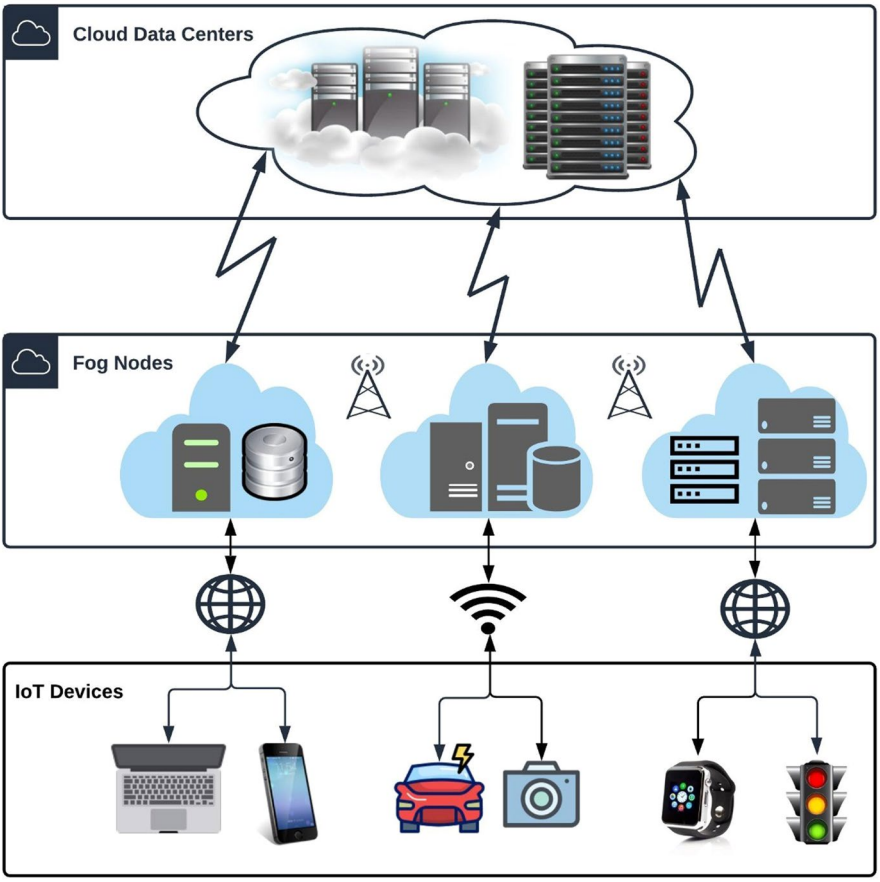


Fig. 1 Three-tier cloud–fog architecture

virtualization technology is usually employed in cloud–fog systems to provide the resources of cloud and fog nodes in the form of VMs. It eliminates server heterogeneity, achieves server consolidation, and improves resource utilization rates [11]. A virtualized cloud–fog system can be efficiently used to deploy bag-of-tasks (BoT) applications such as massive searches, computational biology, and video encoding/decoding. Such applications consist of a number of independent tasks that can run in parallel. However, task scheduling in a cloud–fog system in such a way that achieves the applications' requirements is a challenge. Task scheduling algorithms attempt to find the best assignment of tasks to the available VMs, which optimizes the scheduling objectives. Several scheduling objectives have been targeted, including optimizing energy consumption, maximizing the profits of resource providers, minimizing makespan values, minimizing execution costs, and meeting the deadlines of real-time tasks [9]. Task scheduling in a cloud–fog system is an NP-hard optimization problem. Hence, many metaheuristics, such as the bees life algorithm (BLA), genetic algorithm (GA), and moth-flame optimization (MFO) algorithm, have been employed to provide efficient schedules within a reasonable amount of time. However, these works either consider the static task scheduling problem or ignore the deadline constraint of real-time tasks of IoT applications. Static scheduling algorithms are suitable for small- or medium-scale computing environments, but they are inapplicable to dynamic IoT environments. Hence, more research efforts are needed to propose task scheduling algorithms that consider real-time tasks that have different arrival times. The previous narration motivated us to develop a semidynamic real-time task scheduling algorithm for delay-sensitive IoT applications.

Real-time task scheduling in a cloud–fog system can be formulated as a permutation-based optimization problem. The order of real-time tasks has a significant impact on the performance of the scheduling algorithm. This study presents a semidynamic soft real-time task scheduling algorithm for cloud–fog systems. It employs a permutation-based genetic algorithm called the improved genetic algorithm for permutation-based optimization problems (IGA-POP) [12]. The goal of the IGA-POP is to suggest different permutations for the real-time tasks submitted by IoT end users. Then, the tasks are assigned, in the suggested order, to a VM that provides the minimum expected finish time. The proposed scheduling algorithm is periodically executed every fixed time (τ) to schedule the set of tasks arriving since the last scheduling round. Additionally, the proposed scheduling algorithm fulfills the requirements of various users by achieving a good trade-off between the makespan and total execution cost, where some users prefer to give higher priority to the execution time and others prefer to execute their tasks within a tight budget. The contributions of this study are:

- Suggesting a permutation-based, semidynamic, soft real-time task scheduling algorithm for cloud–fog systems based on the IGA-POP.
- Achieving a good balance between the makespan and the total execution cost while meeting deadline constraints.
- Evaluating the performance of the proposed scheduling algorithm on several datasets against the first fit (FF) algorithm, the best fit (BF) algorithm, the bees

life algorithm (BLA), and the genetic algorithm (GA) in terms of the makespan, total execution cost, failure rate, and average delay time.

The remaining sections of this paper are organized as follows: Sect. 2 summarizes the relevant research works. Section 3 presents the problem formulation of real-time task scheduling in virtualized cloud–fog systems. Section 4 describes the proposed task scheduling algorithm. Section 5 introduces the experiments and performance evaluation. Finally, the paper is concluded in Sect. 6.

2 Literature Review

Task scheduling in a cloud–fog computing environment influences the performance of many cloud–fog-based IoT applications. Recently, several approaches have been proposed to address this challenge. In this section, we cover the task scheduling techniques proposed for cloud, fog, and cloud–fog computing.

2.1 Task Scheduling in the Cloud Computing Environment

Kimpan and Kruekaew presented a hybrid cloud-based task scheduling approach that combines heuristic algorithms with the artificial bee colony (ABC) algorithm. The proposed approach attempts to optimize the makespan and the workload balance among VMs. The performance of the proposed approach is evaluated on different datasets with different numbers of tasks and a fixed number of VMs. The experimental results show that the proposed approach minimizes the makespan with an increasing number of tasks, but it does not give satisfactory results in some cases [13].

Abdullahi and Ngadi employed a discrete version of the symbiotic organism search (SOS) algorithm for task scheduling in a cloud computing environment. The proposed algorithm attempts to enhance the fitness function quality by mimicking mutualism, commensalism and parasitic relationships. It is assessed in terms of the makespan, response time, and degree of imbalance using four different dataset categories. Experiments reveal that the performance of the proposed algorithm improves with an enlarged search space. However, the approach is only compared with variants of the particle swarm optimization (PSO) algorithm [14].

Mishra et al. suggested a task scheduling approach for cloud computing based on ant colony optimization (ACO). The proposed algorithm aims to reduce the makespan and the average waiting time. The allocation of VMs is performed with a probability function. The performance of the proposed algorithm is compared to that of the random and round robin algorithms. The experiments reveal that the proposed algorithm has satisfactory performance. However, the proposed algorithm is not evaluated against other metaheuristics [15].

Gill et al. proposed a cloud resource provisioning and scheduling approach (BULLET) based on Particle Swarm Optimization (PSO). The proposed algorithm uses various Quality of Service (QoS) parameters such as availability, resource

utilization, latency, and reliability. The conducted experiments revealed the superiority of BULLET against other cloud PSO-based task scheduling algorithms. However, BULLET was designed to work in the cloud environment without the ability to consider the fog resources. In addition, it does not consider the deadline constraints of real-time tasks [16].

Natesan and Chokkalingam introduced a task scheduling approach for cloud computing based on enhanced grey wolf optimization (EGWO). The proposed approach minimizes the makespan and the energy consumption by modifying the hunting and encircling behavior of the basic GWO algorithm. It uses a resource information server to determine the available resources of VMs to select the most appropriate VM for each request. However, their approach does not consider the different arrival times of tasks. Moreover, it does not leverage the capabilities of fog computing [17].

Reddy and Kumar proposed a multiobjective task scheduling approach in a cloud environment based on modified ant colony optimization (MACO). The proposed algorithm attempts to optimize the makespan and the load imbalance degree. The conducted experiments show the ability of the proposed approach to significantly decrease the makespan and the imbalance degree compared to the basic ACO algorithm. However, this study ignores the cost parameters. Additionally, it does not leverage the fog nodes to reduce the processing time of the submitted tasks. Moreover, a detailed analysis of its findings is lacking [18].

Sangaiah et al. introduced a resource allocation algorithm in the IoT environment based on the whale optimization algorithm (WOA). The proposed algorithm aims to find the optimal schedule and to minimize the total communication cost between resources and gateways. The performance of the proposed algorithm is evaluated on several benchmarks against two other algorithms. The obtained results revealed the superiority of the proposed algorithm with respect to the total communication cost. However, this algorithm only addresses the static task scheduling problem in a cloud environment. In addition, it ignores deadline constraints of real-time tasks [19].

2.2 Task Scheduling in the Fog Computing Environment

Bitam et al. suggested a bioinspired task scheduling approach for fog computing based on the bees life algorithm (BLA). The proposed approach depends on two main behaviors (food foraging and reproduction) of bees. Experimental studies show that the BLA has a lower scheduling cost than the PSO algorithm and GA. In addition, it delivers reduced execution times and memory consumption. However, the proposed approach does not consider deadline-constrained tasks [20].

Mishra et al. proposed a metaheuristic-based task scheduling technique for industrial applications in a fog computing environment. The proposed technique aims to optimize the makespan and energy consumption trade-off. Additionally, it attempts to address the heterogeneity problem of fog nodes. Three metaheuristics are used to perform the task allocation, namely, the PSO algorithm, the binary PSO (BPSO) algorithm, and the bat algorithm (BA). The conducted experiments show that the BA outperforms the two other algorithms. However, the proposed approach is not applicable for dynamic scheduling [10].

Wan et al. proposed a load balancing scheduling approach for multitask and multi-object systems in a fog environment. The proposed approach aims to minimize the energy consumption in smart factories and to maximize the resource utilization of fog nodes. First, the proposed approach establishes an energy consumption model and formulates the optimization function. Then, an optimal solution is found through an improved version of the PSO algorithm. Finally, a multiagent system performs dynamic scheduling of manufacturing clusters. The simulation results show that the proposed framework reduces task delays in fog nodes. However, their study ignores the execution cost factor [21].

Yang et al. presented a low-complexity, energy-efficient task scheduling algorithm in a fog environment. The proposed algorithm introduces the concept of helper nodes that can execute tasks offloaded from the same task node. The algorithm aims to find the optimal scheduling decision for both the task node and the neighboring helper nodes. Simulation results reveal that the proposed algorithm outperforms traditional task scheduling approaches in terms of energy efficiency (EE). However, the experiments do not consider makespan and cost as evaluation parameters [22].

Rahbari et al. presented a task scheduling approach in a fog environment based on a greedy knapsack algorithm. The proposed approach is assessed on two case studies (EEG beam tractor and intelligent surveillance) in terms of the execution cost, network usage, and energy consumption. The experiments reveal that the proposed approach outperforms the FCFS, concurrent, and delay-priority algorithms in both case studies. However, the performance of the proposed algorithm is not evaluated against other metaheuristics [23].

Li et al. presented a hybrid task scheduling algorithm for a fog computing environment that aims to optimize user satisfaction. The proposed scheduling algorithm combines the C-Means clustering algorithm and the PSO algorithm. First, the resources are clustered to reduce the search scale of resources. Then, the resources are allocated to the tasks based on the matching weight. The performance of the proposed algorithm is evaluated against the max–min algorithm. The obtained results show the superiority of the proposed algorithm in terms of user satisfaction. However, the proposed algorithm ignores many important metrics, such as the makespan and the execution cost. Additionally, the performance of the proposed algorithm is not evaluated against other metaheuristics [5].

Chen et al. introduced an online task scheduling algorithm for soft real-time systems in the mobile edge computing environment. The conducted experiments show that the proposed heuristic algorithm is better than classic scheduling algorithms including first come first served (FCFS) algorithm, earliest due date (EDD) scheduling algorithm, and nearest assignment strategy. However, the proposed heuristic algorithm only focuses on the edge resources and neglects the gained benefits of using the cloud resources as a fallback solution [24].

Ghobaei-Arani et al. proposed a task scheduling algorithm based on moth-flame optimization (MFO) for a fog computing environment. The proposed algorithm aims to minimize the task execution time and the task transfer time. The performance of the proposed algorithm is compared to the PSO algorithm, a multiobjective version of the GA called NSGA-II, and the BLA. The obtained results reveal the superiority

of the proposed algorithm over the other algorithms. However, the proposed algorithm is restricted to static task scheduling [11].

2.3 Task Scheduling in the Cloud–Fog Computing Environment

Deng et al. proposed an approximate approach for workload assignment in a fog-cloud environment to minimize power consumption and task latency. The proposed approach decomposes the main problem into three independent subproblems. First, convex optimization is used to find the optimal correlation between power consumption and latency in fog computing. Second, a nonlinear integer method is used to find the optimal correlation between power consumption and latency in cloud computing. Finally, the Hungarian method is used to minimize the transmission latency from the fog server to the cloud server. The conducted experiments show that the proposed approach saves communication bandwidth and reduces the transmission latency. However, this method is less suitable for fog computing infrastructure because the cloud hub is responsible for work allocation [25].

Pham et al. proposed a heuristic-based workflow scheduling algorithm for cloud–fog computing environments. The proposed algorithm attempts to balance the makespan and execution cost while meeting the deadline constraint. It employs a task reassignment strategy that depends on the critical path of the directed acyclic graph to refine the schedules suggested by the proposed algorithm. The experimental results show that the proposed algorithm achieved the best results in terms of the makespan and the execution cost compared to other heuristics. Additionally, it achieves reasonable performance in terms of schedule length. However, the proposed algorithm assumes that all tasks have the same arrival time [4].

Kamal et al. presented a load balancing scheduling approach for cloud–fog computing to solve a constraint satisfaction problem (CSP). First, the tasks are checked against assignment conflicts. Then, they are randomly assigned to VMs. The approach is compared to the throttled and round robin algorithms in terms of the processing time, response time, and cost. The proposed approach provides better simulation results by allocating optimal resources to requests. However, the performance improvement is insignificant compared to the other techniques [26].

Mahmoud et al. introduced an energy-aware task scheduling algorithm in a cloud–fog environment. The proposed scheduling algorithm uses the edge-wards placement strategy, which assigns tasks to fog and cloud nodes based on the remaining CPU capacity and energy consumption. The performance of the proposed algorithm is assessed under different scenarios (cloud, fog, and cloud–fog) in terms of energy consumption, end-to-end average latency, and average network usage. The obtained results reveal the superiority of the proposed algorithm over other heuristics in terms of the performance metrics used. However, important performance criteria, such as the makespan and execution cost, are not considered [8].

Nguyen et al. proposed a task scheduling algorithm in a cloud–fog environment based on the GA. The proposed algorithm aims to balance the makespan and the monetary cost. The experiments are performed on different datasets with varying sizes in different scenarios. The proposed algorithm outperforms the BLA, the

modified PSO (MPSO) algorithm and the round robin (RR) algorithms in both fog and cloud–fog environments. However, the proposed algorithm only addresses the static scheduling problem. Additionally, it ignores important constraints such as budget, deadline, and resource limitations [2].

Boveiri et al. presented a workflow scheduling algorithm for cloud–fog environments based on the max–min ant system (MMAS). The proposed algorithm aims to consider the priorities of tasks in a proper manner. Several task graphs with different shape parameters are used to assess the performance of the proposed algorithm. The obtained results show the superiority of the proposed algorithm in terms of the normalized schedule length over many heuristics. However, the proposed approach does not consider tasks that have different arrival times [27].

Abdelmoneem et al. introduced a mobility-aware scheduling approach for healthcare tasks in an IoT environment. The proposed approach employs the weighted sum method (WSM) and an upgraded version of the modified BAlance-reduced (MBAR) scheduling and allocation algorithm. It attempts to balance the computational load among cloud and fog devices. In addition, it aims to reduce the makespan, energy consumption, and number of missed tasks. However, the proposed approach ignores the total execution cost, which is an important factor for many users. Moreover, it ignores the deadline constraint and only classifies the tasks into one of five classes based on their criticality [28].

Fellir et al. proposed a multiagent-based task scheduling algorithm for prioritized dependent tasks in the cloud–fog environment. The conducted experiments show the superiority of the proposed algorithm against the FCFS method in terms of energy consumption and total execution cost. However, more experiments are needed to validate the algorithm under different scenarios against more sophisticated heuristics and metaheuristics. Moreover, the proposed approach does not consider the delay sensitivity of the majority of IoT applications [29].

Nikoui et al. introduced a genetic-based cost-aware task scheduling algorithm for cloud–fog environments. The proposed algorithm aims to find the best schedule that optimizes the execution cost of hard real-time applications. The simulation results show that the proposed algorithm is better than the round robin (RR) algorithm. However, more experiments on larger datasets are needed to validate the performance of the proposed algorithm under more difficult scenarios. In addition, the performance of the proposed algorithm should be compared to other real-time heuristic and metaheuristic scheduling algorithms to prove its worthiness in this category of scheduling algorithms [30].

Baniata et al. presented a blockchain-assisted task scheduling algorithm in the cloud–fog environment based on the ant colony optimization (ACO) algorithm. The proposed approach uses blockchain miners to generate efficient task schedules and rewards miner nodes for their participation in producing the best schedule. The conducted experiments show that the proposed algorithm has noticeable enhancements with respect to privacy, total execution time, and network load. However, the proposed algorithm is a static scheduling algorithm that assumes all tasks have the same arrival time [31].

Singh and Singh introduced an energy-efficient delay-aware task scheduling algorithm in the cloud–fog environment based on Levy-flight moth flame optimization

(LMFO) algorithm. The simulation results reveal that the proposed algorithm is better than other recent scheduling algorithms in terms of several performance metrics. However, the number of nodes used in the simulation is very small and the proposed algorithm should be validated using lager datasets under more realistic scenarios [32].

The differences between our proposed algorithm and the works proposed in the literature can be summarized as follows: First, the proposed algorithm applies a semidynamic scheduling approach in which the tasks have different arrival times. Hence, the proposed algorithm is repeatedly executed every fixed time at the fog broker node to schedule the batch of tasks arriving since the last execution. In contrast, the scheduling algorithms proposed in [2, 10, 11] apply a static scheduling approach. Additionally, the algorithms proposed in [4, 16, 25, 28] assume that all tasks have the same arrival time. Second, the proposed scheduling considers the total execution time and the total cost as performance metrics. In addition, it attempts to fulfill the deadline constraint. In contrast, works proposed in [5, 8, 18, 21] ignore the makespan and/or execution cost metrics. Moreover, the works proposed in [2, 17, 19, 26, 27] ignore the deadline constraints of many IoT applications. Finally, the performance of the proposed scheduling algorithm is evaluated against traditional heuristic and metaheuristic approaches, including FF, BF, the GA, and the BLA. The experimental results show the superiority of the proposed algorithm against the other algorithms. Moreover, the proposed algorithm achieves a good balance between the makespan and the total execution cost. In contrast, the works proposed in [5, 15, 22] did not compare their approaches to other metaheuristics. Additionally, the results provided by [14] are only compared to the PSO algorithm. Additionally, the findings in [13, 24] do not reveal a significant improvement compared to other techniques.

Table 1 presents a brief comparative study on the reviewed scheduling algorithms and the proposed scheduling algorithm.

3 Real-Time Task Scheduling in Cloud–Fog Computing

3.1 Cloud–Fog System Model

Many IoT applications, such as healthcare and smart traffic, are real-time bag-of-tasks (BoT) applications that can be decomposed into several independent real-time tasks. These applications generate huge amounts of data that cannot be processed by IoT devices. Hence, cloud–fog computing is used to address the inherent limitations of IoT devices.

Cloud–fog computing can be implemented using traditional ways such as virtual machines or using docker containers. Both techniques are based on the concept of virtualization to provide the independency on the underlying hardware. VMs provide better separation between applications and operation system. Also, they enable the concept of process migrations at run-time which reduces the power consumption and improves the load balancing. On the other hand, in VMs, there exists a single hypervisor which introduces a single point of failure.

Table 1 A summary for the task scheduling techniques

References/year	Scheduling	Task	Environment	Algorithm(s)	Performance measures	Deadline
[13]/(2016)	Static	Independent	Cloud	Hybrid: heuristic algorithms + ABC	Makespan	✖
[14]/(2016)	Static	Independent	Cloud	SOS PSO	Makespan Response time Imbalance degree	✖
[15]/(2017)	Static	Independent	Cloud	ACO	Makespan Average waiting time	✖
[16]/(2018)	Static	Independent	Cloud	PSO	Availability Resource utilization Latency Reliability	✖
[18]/(2019)	Static	Independent	Cloud	Enhanced GWO	Makespan Energy consumption	✖
[17]/(2019)	Static	Independent	Cloud	Modified ACO	Makespan Imbalance degree	✖
[19]/(2020)	Static	Independent	Cloud	WOA	Total communication cost	✖
[20]/(2018)	Static	Independent	Fog	BLA	CPU execution time Memory consumption	✖
[10]/(2018)	Static	Independent	Fog	PSO Binary PSO BAT	makespan Energy consumption	✖
[21]/(2018)	Static	Dependent	Fog	Improved PSO	Workload ratio Robot mean difference	✖
[22]/(2018)	Static	Dependent	Fog	TDMA + MEETS	Energy efficiency	✖
[23]/(2019)	Static	Dependent	Fog	Greedy Knapsack Algorithm	Energy consumption Execution cost Sensor lifetime	✖
[5]/(2019)	Static	Dependent	Fog	Hybrid: C-Means Clustering + PSO	User satisfaction	✖
[24]/(2019)	Dynamic	Independent	Fog	Heuristic-based	Total penalty of tardiness	✓

Table 1 (continued)

References/year	Scheduling	Task	Environment	Algorithm(s)	Performance measures	Deadline
[11]/(2020)	Static	Independent	Fog	Moth-Flame Optimization	Task execution time Task transfer time	✖
[25]/(2016)	Static	Independent	Cloud–fog	Hybrid: convex optimization + nonlinear integer method + Hungarian method	Power consumption Transmission delay	✖
[4]/(2017)	Static	Dependent	Cloud–Fog	Heuristic-based	Makespan Execution cost	✓
[26]/(2018)	Static	Independent	Cloud–fog	Min-conflicts	Execution cost Processing time Response time	✖
[8]/(2018)	Static	Dependent	Cloud–fog	Heuristic-based	Energy consumption Average latency Average network usage	✖
[2]/(2019)	Static	Independent	Cloud–fog	GA	Makespan Execution cost	✖
[27]/(2019)	Static	Dependent	Cloud–fog	Max–Min Ant System	Normalized schedule Length	✖
[28]/(2020)	Static	Independent	Cloud–fog	WSM MBAR	Makespan Network load Energy consumption Number of missed tasks	✖
[29]/(2020)	Static	Dependent	Cloud–fog	Multi-Agent Model	Energy consumption Total execution cost	✖
[30]/(2020)		Independent	Cloud–fog	GA	Success rate Execution cost	✓
[28]/(2021)	Static	Independent	Cloud–fog	ACO	Privacy Network load Total execution	✖

Table 1 (continued)

References/year	Scheduling	Task	Environment	Algorithm(s)	Performance measures	Deadline
[32]/(2022)	Static	Independent	Cloud–fog	LMFO	Energy consumption Computation delay CO Emission temperature Emission	✖
Proposed	Semi-dynamic	Independent	Cloud–fog	IGA-POP	Makespan Execution cost Failure rate Average delay time	✓

Containers are more lightweight virtualization concept. They are lightweight in terms of booting time and resource needs because they depend on the host operating system. They allow better portability and interoperability. Additionally, they are highly scalable compared to VMs. On the other hand, containers are more application-specific compared to VMs. Containers focus on the applications and their dependencies. They are preferred if the target is to provide high availability and scalability, while VMs provide more flexibility and it is needed if the requirement is to create a more secure system.VM provides better IaaS solution, while containers provide better SaaS solution. We can conclude that, both VMs and Containers can be used for cloud–fog implementation and selecting one of them depends on the user requirements [33]. Despite the advantage of containers in terms of its ability to encapsulate, deploy, and isolate applications, lightweight operations, as well as efficiency and flexibility in resource sharing. The implementation of this study is based on VMs to investigate the performance of the proposed algorithm in a highly constrained/ challengeable environment. Hence, the computing resources are presented the form of VMs that have different computing capabilities.

Real-time task scheduling in virtualized cloud–fog computing aims to find the optimal assignment of real-time tasks submitted by IoT end users to available VMs. As shown in Fig. 2, the cloud–fog system consists of f virtualized fog nodes, c virtualized cloud nodes, a fog broker, and a cloud broker.

Each broker contains three main components: a task scheduler, a resource monitor, and a task monitor. The task scheduler applies the task scheduling algorithm that achieves the adopted scheduling policy. The resource monitor tracks the available capacity of the different VMs. The task monitor tracks the progress of task execution and returns the execution results to IoT users. The following assumptions are made about the cloud–fog model:

The tasks submitted by IoT end users have different arrival times and deadline times.

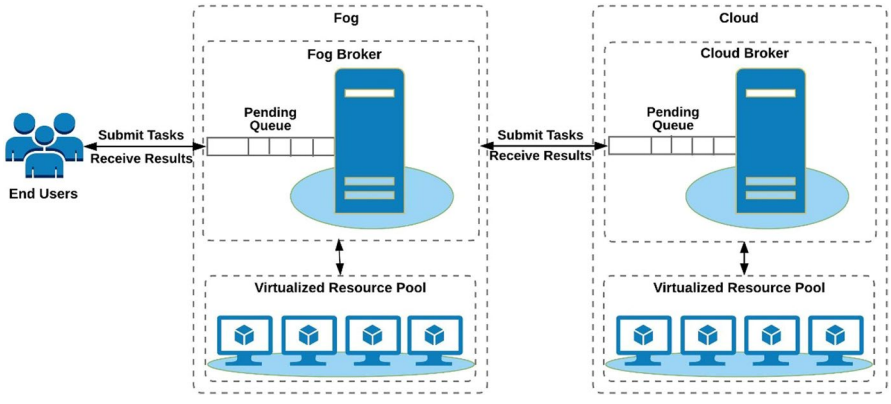


Fig. 2 Task scheduling in cloud–fog system

The scheduling algorithm is periodically executed each fixed time (τ) to schedule the set of tasks arriving at the pending queue of the broker since the last scheduling round.

Figure 3 shows the sequence diagram of the task scheduling operation in a cloud–fog system. First, all requests are forwarded immediately from IoT devices to the fog broker. Then, the fog broker assigns the real-time tasks to the fog nodes, collects execution results, and sends the results back to the IoT devices. However, if the fog nodes are saturated, the fog broker immediately forwards the tasks to the cloud broker. The cloud broker assigns the arrived real-time tasks to the cloud nodes, collects execution results, and sends the results back to the fog broker, who in turn sends the results back to the IoT devices.

3.2 Problem Formulation

This study considers a cloud–fog system that consists of f heterogeneous virtualized fog nodes $[FN_1, FN_2, \ldots, FN_f]$, c heterogeneous virtualized cloud nodes $[CN_1, CN_2, \ldots, CN_c]$, and N independent real-time tasks $[RT_1, RT_2, \ldots, RT_N]$. Additionally, the virtualized nodes host M heterogeneous VMs $[VM_1, VM_2, \ldots, VM_M]$. The notations used by the proposed scheduling algorithm are shown in Table 2.

The semidynamic real-time task scheduler aims to assign each submitted task to a virtual machine. Each VM can be assigned one or more tasks. The list of tasks assigned to VM_j is denoted by $[RT^j_a, RT^j_b, \ldots, RT^j_{N_j}]$. Therefore, the proposed schedule is repre-

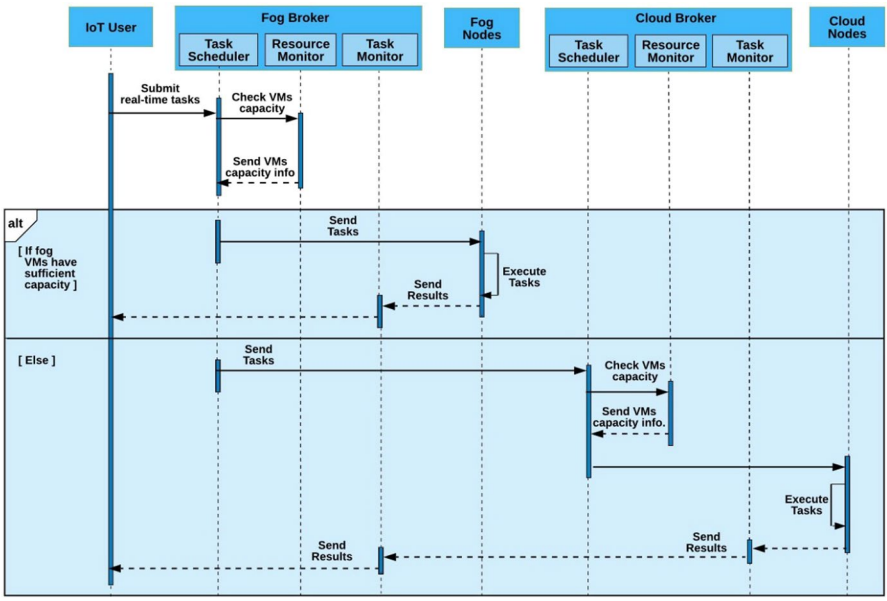


Fig. 3 A sequence diagram of task scheduling operation in a cloud–fog system

Table 2 Notations used by the proposed scheduling algorithm

Notation	The meaning of the notation
N	The number of real-time tasks
RT_i	The ith real-time task
LRT_i	The length of the RT_i (MI)
IFS_i	The input file size of the RT_i (MB)
OFS_i	The output file size of the RT_i in (MB)
BRT_i	The bandwidth requirement of the RT_i
MRT_i	The memory requirement of the RT_i
BAT_i	The broker arrival time of the RT_i
NAT_i	The node arrival time of the RT_i
DRT_i	The deadline of the RT_i
PT_i	The pending time of the RT_i
M	The number of VMs
VM_j	The jth virtual machine
Pe_Num_j	The number of computing elements in the VM_j
Pe_Mips_j	The computing power of a computing element in the VM_j (MIPS)
RAM_j	The memory capacity of the VM_j
BW_j	The bandwidth of the VM_j
N_j	The number of tasks assigned to the VM_j
PC_j	The processing cost per time unit of the VM_j
MC_j	The memory cost per storage unit of the VM_j
BC_j	The bandwidth cost per data unit of the VM_j
WT_{ij}	The waiting time of the RT_i on the VM_j
EET_{ij}	The expected execution time of the RT_i on the VM_j
EFT_{ij}	The expected finish time of the RT_i on the VM_j
$Cost_{ij}^{CPU}$	The CPU usage cost of executing the RT_i on the VM_j
$Cost_{ij}^{RAM}$	The memory usage cost of executing the RT_i on the VM_j
$Cost_{ij}^{BW}$	The bandwidth usage cost of executing the RT_i on the VM_j
$Cost_{ij}$	The total cost of executing the RT_i on the VM_j

sented by $[RT_1^j, \ldots, RT_n^k, \ldots, RT_N^M]$. The expected finish time of $\mathrm{RT_i}$ on $\mathrm{VM_j}$ is denoted by $\mathrm{EFT_{ij}}$ and is calculated by Eq. (1).

$$EFT_{ij} = BAT_i + PT_i + WT_{ij} + EET_{ij} \tag{1}$$

where EFT_{ij} is the expected finish time of RT_i on VM_j; BAT_i is the broker arrival time of, RT_i, which denotes the arrival time of the task at the pending queue of the broker; PT_i is the pending time of, RT_i, which denotes the amount of time the task waits at the pending queue before being scheduled to a VM_j; WT_{ij} is the waiting time of, RT_i, which denotes the amount of time the task waits at the waiting queue of VM_j before being processed; and EET_{ij} is the expected execution time of $\mathrm{RT_i}$ on $\mathrm{VM_j}$. The pending time PT_i of RT_i is calculated by Eq. (2).

$$PT_i = NAT_i - BAT_i \tag{2}$$

where NAT_i is the node arrival time of, RT_i, which denotes the arrival time of the task at the selected VM. The expected execution time EET_{ij} of RT_i on $\mathrm{VM_j}$ is calculated by Eq. (3).

$$EET_{ij} = \frac{LRT_i}{Pe\_Num_j * Pe\_Mips_j} + \frac{IFS_i + OFS_i}{BW_j} \tag{3}$$

where LRT_i is the task length in *million instructions* (*MI*), IFS_i is the input file size of, RT_i, OFS_i is the output file size of, RT_i, Pe_Num_j is the number of processing elements of, VM_j, Pe_Mips_j is the computing power of each computing element in $\mathrm{VM_j}$ in *million instructions per second* (*MIPS*), and BW_j is the bandwidth of.VM_j. If $EFT_{ij} \leq DRT_i$, then RT_i is marked as "*passed*"; otherwise, it is marked as "*failed*". The failure rate is calculated by Eq. (4).

$$Failure\ Rate = \frac{No.\ of\ Failed\ Tasks}{N} \tag{4}$$

where N is the total number of tasks.

Assume that RT_l^j is the last task assigned to $\mathrm{VM_j}$ and $ExeTime_j$ is the total time needed to finish all the tasks assigned to.VM_j. $ExeTime_j$ is calculated by Eq. (5).

$$ExeTime_j = EFT_{lj} \tag{5}$$

where EFT_{lj} is the expected finish time of the last task assigned to.$\mathrm{VM_j}$. *Makespan* is the total time needed by the system to finish all the tasks. It is calculated by Eq. (6), where M is the total number of VMs.

$$Makespan = \underset{1 \leq j \leq M}{Max} \left[ExeTime_j\right] \tag{6}$$

Let *MinMakespan* be the minimum time needed by the system to finish all the tasks (i.e., it is the lower bound *Makespan*). It can be obtained when all the VMs finish their assigned tasks at the same time. It is calculated for a semidynamic environment that runs the task scheduler for each fixed time (τ) as follows:

$$MinMakespan = \begin{cases} \frac{\sum_i^N LRT_i}{\sum_j^M Pe\_Num_j * Pe\_Mips_j} + \frac{\sum_{i=1}^N (IFS_i + OFS_i)}{\sum_j^M BW_j}, if\ k = 1 \\ \left(\frac{((k-1)*\tau)+(k*\tau)}{2}\right) + \frac{\sum_i^N LRT_i}{\sum_j^M Pe_{Num_j} * Pe_{Mips_j}} + \frac{\sum_{i=1}^N (IFS_i + OFS_i)}{\sum_j^M BW_j}, if\ k > 1 \end{cases} \tag{7}$$

where k ($k \in [1, \infty]$) represents the current scheduling round, N is the total number of tasks arriving from the 1st scheduling round to the kth scheduling round, and M is the total number of VMs. The first scheduling round (i.e., k = 1) starts at time 0 for all tasks whose broker arrival times equal 0. The first part of the second case in Eq. (7) addresses the different broker arrival times of real-time tasks in subsequent

scheduling rounds. It represents the average broker arrival times of the real-time tasks for the current scheduling round.

Therefore, the tasks' broker arrival times are $\leq (k-1) * \tau$ and the scheduling times are $(k-1) * \tau$ for the subsequent scheduling rounds (i.e., $k > 1$).

Task execution in cloud–fog systems involves a monetary cost that must be paid, including the processing cost, memory cost, and bandwidth usage cost. In the proposed model, we have adopted a pricing model similar to the Alibaba cloud resource pricing model in which each resource has a separate pricing model. For example, bandwidth consumption has a cost which differs from memory usage or computing power usage [34]. Let PC_j be the processing cost per time unit of, VM_j, MC_j be the memory cost per storage unit of, VM_j, BC_j be the bandwidth cost per data unit of, VM_j, MRT_i be the memory requirement of, RT_i, and BRT_i be the bandwidth requirement of.RT_i. The cost of executing RT_i on VM_j is calculated as follows:

$$Cost_{ij} = Cost_{ij}^{CPU} + Cost_{ij}^{RAM} + Cost_{ij}^{BW} \tag{8}$$

$$Cost_{ij}^{CPU} = PC_j * EET_{ij} \tag{9}$$

$$Cost_{ij}^{RAM} = MC_j * MRT_i \tag{10}$$

$$Cost_{ij}^{BW} = BC_j * BRT_i \tag{11}$$

where $Cost_{ij}^{CPU}$ is the CPU usage cost of executing RT_i on, VM_j, $Cost_{ij}^{RAM}$ is the memory usage cost of executing RT_i on, VM_j, $Cost_{ij}^{BW}$ is the bandwidth usage cost of executing RT_i on, VM_j, and $Cost_{ij}$ is the cost of executing RT_i on.VM_j. The bandwidth requirement BRT_i of RT_i is calculated by Eq. (12).

$$BRT_i = IFS_i + OFS_i \tag{12}$$

where IFS_i is the input file size of RT_i and OFS_i is the output file size of RT_i. The total cost of executing all the tasks in the cloud–fog system is calculated as follows:

$$TotalExecutionCost = \sum_{i=1}^{N} \sum_{J=1}^{M} x_{ij} * Cost_{ij} \tag{13}$$

where x_{ij} is a binary variable equal to 1 if RT_i is assigned to VM_j and 0 otherwise.

Let *MinExecutionCost* be the lowest cost needed to execute all the tasks in the cloud–fog system. Given the cost information of the different VMs, it is obtained by executing the tasks on the cheapest node. Hence, the minimum cost $MinCost_i$ needed to execute RT_i is calculated as follows:

$$MinCost_i = \underset{1 \leq j \leq M}{Min} [Cost_{ij}] \tag{14}$$

$$MinExecutionCost = \sum_{i=1}^{N} MinCost_i \tag{15}$$

From the above description, semidynamic real-time task scheduling is formulated as an optimization problem defined by Eq. (16).

$$Maximize\left(\frac{MinMakespan}{Makespan}, \frac{MinExecutionCost}{TotalExecutionCost}\right) \tag{16}$$

In the most optimistic scenario, *Makespan* is equal to *MinMakespan* and *TotalExecutionCost* is equal to *MinExecutionCost*. Hence, the upper bound of each term in Eq. (16) is equal to 1.

4 IGA-POP Based Real-Time Task Scheduling in the Cloud–Fog Environment

In this section, a permutation-based, semidynamic, real-time task scheduling algorithm is presented based on the IGA-POP [12]. The goal of the IGA-POP is to provide different permutations for the real-time tasks in the current scheduling round. Given the suggested permutation, the real-time tasks are scheduled, in the submitted order, to the VM that gives the minimum expected finish time. The pseudocode of the proposed real-time task scheduling is given in Algorithm 1.

Algorithm 1: The proposed Scheduling algorithm

Input: Number of scheduling rounds (nr), a list of real-time tasks (l_{RT}), a list of VMs (l_{VM}), population size(p), and maximum number of iterations (t_{max}).
Output: The cumulative task schedule, S_c.

```
1   S_c = φ.                                          /*Initially an empty schedule */
2   For k = 1 to nr
        // the first scheduling round
3       If (k = 1) Then
4           Create a list of real-time tasks (l_RT^k) that have BAT = 0.
            // schedule the current list of real-time tasks using IGA-POP algorithm.
5           S_k = IGA-POP (p, t_max, l_RT^k, l_VM).          /*Algorithm 2*/
        // the subsequent scheduling rounds.
6       Else
7           Create a list of real-time tasks (l_RT^k) that have ((k − 2) * τ) < BAT ≤ ((k − 1) * τ).
            // schedule the current list of real-time tasks using IGA-POP algorithm.
8           S_k = IGA-POP (p, t_max, l_RT^k, l_VM).          /*Algorithm 2*/
9       End
10      Update the current resources of the available VMs.
11      S_c = S_c ∪ S_k.
12  End
13  Return the cumulative task schedule, S_c.
```

Algorithm 1 presents a semidynamic real-time task scheduling algorithm that runs periodically each fixed time τ; i.e., the times of scheduling rounds are $0, \tau, 2\tau, 3\tau, \ldots, etc$. Given a list of real-time tasks l_{RT} and a list of VMs l_{VM}, Algorithm 1 starts with initializing the cumulative schedule S_c. According to their broker arrival times, the real-time tasks are partitioned into sublists l_{RT}^k to be scheduled in the

different scheduling rounds. Each round of the IGA-POP (Algorithm 2) is called to suggest the best schedule S_k for the current set of real-time tasks. Then, the available memory capacity of the VMs is updated by adding the capacity released by the finished tasks. Finally, S_k is merged with the best schedules of the previous scheduling rounds to form the cumulative task schedule S_c, which is eventually returned.

Algorithm 2: IGA-POP algorithm

Input: Population size(p), Number of iterations (t_{max}), the list of current real-time tasks (l^k_{RT}), and the list of VMs (l_{VM}).
Output: The best schedule(S_k).

Create a population Pop_0 that includes p permutations each of length $\left| l^k_{RT} \right|$.
Set the initial values of r_1 and r_2.
For $t = 0$ to $t_{max} - 1$
  For $l = 1$ to p
    //real-time task scheduling based on the minimum expected finish time
    For each real-time task RT_i, $i \in \{1, 2, \ldots, N_k\}$
      For each virtual machine VM_j, $j \in \{1, 2, \ldots, M\}$
        If VM_j has enough capacity for RT_i **Then**
          Calculate the expected finish time of task (EFT_{ij}) of RT_i if it is assigned to VM_j. /* Eq. (1) */
        End
      End
      Assign the RT_i to the VM_i with the minimum expected finish time EFT_{ij}.
    End
    Calculate the utility function for each permutation X in the population Pop_t. /* Eq. (17) */
  End
  // Create the new population
  Determine the B best permutations (Pop^*_t) in the population Pop_t.
  For $l = 1$ to p
    Update r_1.
    If ($r_1 < 0.5$) **Then**
    // Apply the mutation operator
      Apply the mutation operator on X_l to produce the new permutation X'_l.
      Add X'_l to the new population Pop_{t+1}.
    Else
    //Apply the crossover operator
      Pick a permutation X_r randomly from Pop^*_t using the roulette wheel selection algorithm.
      Combine X_l and X_r with a copy ratio r_2 to produce the offspring permutation X_c.
      Calculate the utility function of X_l, X_r, and $X_c$and add the best to the next population Pop_{t+1}.
    End
  End
  Update r_2. /*Eq. (18) */
End
Return the best schedule (S_k) that employs the best permutation (X_{best}) in the last population.

Algorithm 2 is a modified version of the IGA-POP, which was first presented to address permutation-based problems [12]. The goal of the IGA-POP is to provide the best permutations for real-time tasks. For each suggested permutation, the real-time task RT_i is assigned, in the submitted order, to VM_j, which gives the minimum expected finish time EFT_{ij}. Cloud VMs are only used when the fog VMs are saturated. Additionally, the selected VM_j should have enough memory capacity to host RT_i. The objective of the semidynamic real-time task scheduling algorithm is to optimize the total execution time and the total execution cost while meeting the deadline constraint. Hence, the fitness function of each permutation is computed by Eq. (17), which attempts to balance the makespan and the total execution cost trade-off.

$$\textbf{\textit{Maximize}} f(x) = w\left[\frac{MinMakespan}{Makespan}\right] + (1-w)\left[\frac{MinExecutionCost}{TotalExecutionCost}\right] \quad (17)$$

where w ($w \in [0, 1]$) is a weighting parameter that controls the balance between the makespan and the total execution cost. If the makespan and the total execution cost have the same priority, then the value of w is set to 0.5.

To generate the new population, the best permutations (Pop_t^*) in the current population are determined. Then, genetic operators (mutation and ordered crossover) are applied to the current population.r_1 is a uniformly distributed random variable of a value and is bounded by the interval [0, 1]. For each permutation in the current population, if $r_1 < 0.5$, then the permutation is mutated; otherwise, the permutation is crossed over with another permutation selected from Pop_t^* by roulette wheel selection. The proposed algorithm randomly selects one of the following mutation operators: swap, displacement, or reversion [12]. On the other hand, the ordered crossover operation produces a single offspring permutation (X_c) with one part cloned from the parent X_r and the other parts cloned from X_l.r_2 is used to determine the length of the part cloned from the parent X_r. It increases linearly from 0.5 to 1 with the progression of iterations by Eq. (18). In other words, the offspring permutation X_c includes half of the permutation represented by X_r in the first iteration and is an exact copy of X_r in the last iteration. According to the fitness values of X_l,X_r and X_c, the best permutation among them is included in the new population. The optimization process runs until the termination criterion is fulfilled. Finally, the best schedule S_k that uses the best permutation in the final population is returned.

$$r_2 = [1.5 + 0.5 * \left(\frac{2 * t - \left(t_{max} + 1\right)}{t_{max} - 1} \right)]/2 \tag{18}$$

where t is the current iteration and t_{max} is the maximum number of iterations.r_1 and r_2 are used to control the exploration and exploitation of the IGA-POP in the search space.

5 Analysis of the Experiments and Results

In this section, several experiments are conducted to evaluate the performance of the proposed scheduling algorithm against other heuristics and metaheuristics (FF, BF, the BLA, and the GA) in terms of the convergence, makespan, total execution cost, fitness value, elapsed run time, failure rate, and average delay time.

5.1 Experimental Settings

The algorithms used were implemented using MATLAB 9.0.0. All the experiments were performed on the same machine with attributes shown in Table 3.

Cloud and fog nodes have different processing power values measured in *MIPS*, memory capacity values measured in *Megabytes* (*MB*), bandwidth values measured in *Megabytes Per Second* (*MBPS*), and resource usage costs represented in *Grid Dollars* (*G$*). G$ is a currency unit used in the simulation to substitute for real money using a predefined ratio [2, 35]. The virtual charge can be easily mapped to any pricing model

Table 3 Attributes of used machine

Processor	Intel ® Core™ i7, CPU 2.27 GHz
Memory	4 GB
Operating system	Windows 8

of different cloud providers (CPs) such Amazon Web Service (AWS), Microsoft Azure and Google Cloud Platform (GCP). In addition, it gives the ability of the proposed model to be applied in different models with different currencies and in different application domains such as Smart factories, Mobile IoT, Cyber-Physical Systems (CPS) applications. Finally, the most cost-effective algorithm using some virtual currency will be the most cost-effective one when the mapping is done to the real money.

We assume that the VMs hosted in fog nodes have limited resources compared to the VMs hosted in cloud nodes. Additionally, the task execution cost with fog VMs is cheaper than with cloud VMs. Three scenarios were adopted in the conducted experiments regarding the number of VMs used: (10 fog VMs and 5 cloud VMs), (20 fog VMs and 10 cloud VMs), and (30 fog VMs and 15 cloud VMs). The attributes of the VMs are shown in Table 4.

Similarly, the submitted real-time tasks have different task lengths measured in *MI*, memory requirements measured in *MB*, input file sizes measured in *MB*, output file sizes measured in *MB*, arrival times in seconds, and deadline times in seconds. We evaluated our proposed model on six randomly synthetic task datasets with different sizes ranging from 50 to 400 tasks to study the impact of varying number of tasks on the suggested scheduling model. These large datasets are generated with the help of uniform distribution. These tasks represent requests from different IoT devices such as wearable devices, healthcare devices, and smart home sensors etc. The attributes of the real-time tasks are shown in Table 5.

The deadline time DRT_i of a real-time task RT_i is calculated as follows:

$$DRT_i = BAT_i + MaxExeTime_{is} + rand \tag{19}$$

where BAT_i is the broker arrival time of the real-time task RT_i, *rand* is a uniformly distributed random variable bounded by the interval [1:150], and $MaxExeTime_{is}$ is

Table 4 Attributes of the used VMs

Parameter	Fog VM	Cloud VM	Unit
Number of VMs	{10, 20, 30}	{5, 10, 15}	VM
Computing power	[1000:2000]	[3000:5000]	MIPS
RAM	[250:5000]	[5000:20,000]	MB
Bandwidth	[128:1024]	[512:4096]	MBPS
CPU usage cost	[0.2:0.5]	[0.6:1.0]	G$/s
Memory usage cost	[0.01:0.04]	[0.03:0.06]	G$/MB
Bandwidth usage cost	[0.01:0.03]	[0.06:0.1]	G$/MB

Table 5 Attributes of real-time tasks

Parameter	Value	Unit
Task length	[1000:20,000]	MI
Memory	[10:150]	MB
Input file size	[10:100]	MB
Output file size	[10:100]	MB
Broker arrival time	[0:100]	Second
Deadline time	Equation (19)	Second

the time needed to execute the real-time task RT_i on the slowest virtual machine VM_s. The value of $MaxExeTime_{is}$ is calculated as follows:

$$MaxExeTime_{is} = \frac{LRT_i}{Pe\_Num_s * Pe\_Mips_s} + \frac{IFS_i + OFS_i}{BW_s} \tag{20}$$

where the denominator of the first part in Eq. (20) represents the total computing power of VM_s.

The proposed scheduling algorithm is compared to two other metaheuristics: the GA and BLA [36]. In addition, the proposed algorithm is compared to two heuristics: first fit (FF) and best fit (BF). The basic GA and BLA are modified to be applicable for the involved permutation-based optimization. For fair comparisons, the population size is fixed to 200 for all metaheuristics. Table 6 shows the parameter settings of the algorithms used.

5.2 Results and Discussion

The performance of the proposed scheduling algorithm is evaluated with intensive experiments under different scenarios. Ten runs were performed for each metaheuristic, and the average results are presented. Additionally, the heuristics are performed only once.

5.2.1 Convergence Analysis

This section presents the convergence analysis of the proposed scheduling algorithm, the BLA, and the GA. In this experiment, 10 fog VMs and 5 cloud VMs are used to execute a dataset that consists of 100 real-time tasks. The balance coefficient w is set to 0.5. As shown in Fig. 4, the convergence analysis is done in terms of

Table 6 Parameter settings of metaheuristics

Algorithm	Parameter	Value
GA	Crossover rate	0.8
	Mutation rate	0.01
BLA	Neighborhood radius rate	0.75

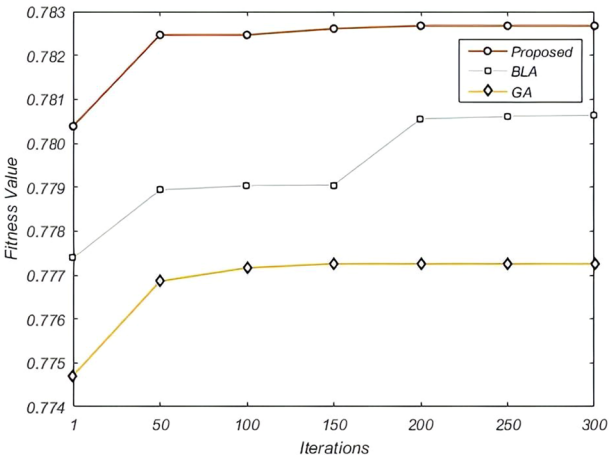


Fig. 4 Convergence of the used metaheuristics

the fitness value, which represents the balance between the makespan and the total execution cost.

Figure 4 shows that the GA and the proposed algorithm converge faster than the BLA. However, the proposed scheduling algorithm has the best fitness value (0.7825) compared with the other metaheuristics. In the BLA, all drones perform a crossover operator with only one queen, which causes the population to easily fall into a local optimum. On the other hand, in the first 100 iterations, the individuals of the proposed algorithm had a wide range of fitness values. Then, the fitness range narrowed, which indicates that the population gathered around the optimum solution. In other words, the proposed algorithm has population diversity; thus, it was able to reach a better solution. For fair comparisons, the maximum number of iterations of the metaheuristics used is fixed to 300 for subsequent experiments.

5.2.2 Balance Coefficient Analysis

This experiment analyzes the effect of the balance coefficient on the makespan and the total execution cost achieved by the proposed scheduling algorithm. In this experiment, 10 fog VMs and 5 cloud VMs are used to execute 100 real-time tasks. The balance coefficient w determines the effect of the makespan and the total execution cost on the optimization process. This experiment investigates different values of w bounded by the interval $[0, 1]$. If $w = 0$, then the proposed scheduling algorithm only uses the total execution cost as an indicator of solution quality during the optimization process. If $w = 1$, then the proposed scheduling algorithm only uses the makespan as an indicator of solution quality during the optimization process. If $w = 0.5$, the proposed algorithm equally depends on the makespan and the total execution cost during the optimization process. The obtained results with different values for the balance coefficient w are shown in Fig. 5.

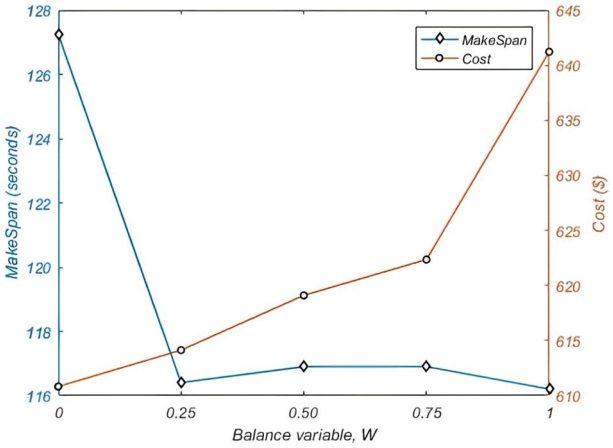


Fig. 5 Effect of the balance coefficient w

Based on Fig. 5, the cost is minimized when $w = 0$; however, the makespan is maximized because the main objective was to use cheap nodes regardless of their computing power. On the other hand, when $w = 1$, the cost is maximized, and the makespan is minimized because the makespan is used only as a guide during the optimization process. Additionally, a good balance is achieved between the makespan and the total execution cost when $w = 0.25$. This experiment shows that the proposed algorithm allows a user to flexibly choose between high-performance execution and cost efficiency by tuning the balance coefficient w. The value of w is fixed at 0.5 in the next experiments to allow the makespan and the execution cost to equally influence the optimization process.

5.2.3 Optimality Analysis

The performance of the different scheduling algorithms is compared to the optimal solution in terms of the fitness value (Eq. 17). The optimal solution is obtained by exploring all possible permutations of the submitted tasks. In this experiment, 3 VMs (2 fog VMs and 1 cloud VM) are used to execute different numbers of tasks ranging from 2 to 10. All the tasks are scheduled in a single scheduling round. The obtained results are shown in Table 7.

Based on Table 7, it is observed that the FF and BF algorithms have the worst performance compared to the optimal solution. The BF algorithm attempts to reduce the resource wastage during task scheduling, while FF assigns the tasks to the first VM that has sufficient resources. Both strategies result in long waiting queues in the active VMs, which means higher makespan values. On the other hand, the BLA and the GA succeed in producing solutions with the same quality as the optimal solution in most cases (7 out of 9 test cases) and produced near-optimal solutions in the remaining cases (2 out of 9 test cases). However, the proposed scheduling algorithm produced the optimal solution in all test cases.

Table 7 Optimality study for the used scheduling algorithms

No. of tasks	Algorithms					
	FF	BF	BLA	GA	Proposed	Optimal
2	0.5144	0.5488	**0.5707**	**0.5707**	**0.5707**	**0.5707**
3	0.5693	0.5415	**0.5933**	**0.5933**	**0.5933**	**0.5933**
4	0.5557	0.5139	**0.6068**	**0.6068**	**0.6068**	**0.6068**
5	0.5466	0.5242	**0.5978**	**0.5978**	**0.5978**	**0.5978**
6	0.5550	0.5314	**0.6088**	**0.6088**	**0.6088**	**0.6088**
7	0.5385	0.5427	**0.6022**	**0.6022**	**0.6022**	**0.6022**
8	0.5369	0.5295	**0.6082**	**0.6082**	**0.6082**	**0.6082**
9	0.5678	0.5379	0.6027	0.6031	**0.6141**	**0.6141**
10	0.5504	0.5188	0.6048	0.6057	**0.6110**	**0.6110**

Bold values indicate the best found results

5.2.4 Fitness Analysis

The fitness function used attempts to balance the makespan and the total execution cost of the real-time tasks. According to the problem formulation, the task scheduling problem is a maximization optimization problem. Hence, the larger the fitness of the algorithm is, the greater the achieved balance between the makespan and the execution cost. This experiment is conducted to investigate the balance achieved by the different scheduling algorithms. Fifteen VMs (10 fog VMs and 5 cloud VMs) are used to execute several task datasets with different sizes. The obtained results are shown in Fig. 6.

Figure 6 demonstrates the superiority of the stochastic algorithms (the GA, BLA, and IGA-POP) against the deterministic algorithms (FF and BF) in terms

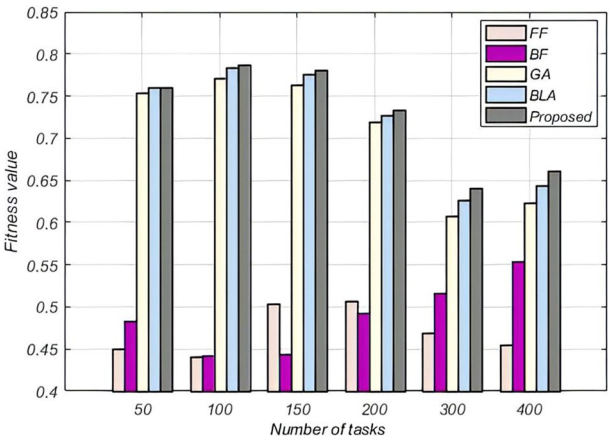


Fig. 6 The fitness of different scheduling algorithms

of the fitness value. Additionally, the fitness decreases with larger datasets for almost all algorithms. However, the proposed scheduling algorithm outperforms the other algorithms in all cases, which reflects the ability of the proposed algorithm to achieve a good balance between the makespan and the total execution cost. Additionally, the stochastic algorithms have very similar performance on the small datasets (50–200 tasks). However, the difference between the proposed scheduling algorithm and the other stochastic algorithms becomes clearer when handling the larger datasets (300–400 tasks). This reflects the ability of the proposed scheduling algorithm to efficiently explore large search spaces due to the balance between the exploration and exploitation achieved by the IGA-POP.

5.2.5 Performance Evaluation

This section investigates the performance of all algorithms under different scenarios (18 scenarios) in terms of the makespan, the failure rate, the average delay time, and the total execution cost. Six task datasets with different sizes (ranging from 50 to 400 tasks) are executed on three different combinations of VMs ((10 fog VMs and 5 cloud VMs), (20 fog VMs and 10 cloud VMs), and (30 fog VMs and 15 cloud VMs)). Table 8 presents the makespan of the different algorithms under different scenarios.

Based on Table 8, it is observed that increasing the number of tasks increases the makespan of all scheduling algorithms for all VM combinations. Additionally, increasing

Table 8 Makespan of different scheduling algorithms under different scenarios (in seconds)

Algorithm	Tasks					
	N = 50	N = 100	N = 150	N = 200	N = 300	N = 400
{10 Fog VMs U 5 VMs Cloud}						
FF	288.9	313.3	308.2	366.3	468.3	482.4
BF	197.9	325.0	423.2	362.3	453.1	455.4
GA	111.3 ± 0.3	118.7 ± 0.8	149.4 ± 1.7	194.5 ± 5.8	358.1 ± 20.6	369.8 ± 17.7
BLA	**109.9 ± 0.1**	116.9 ± 0.3	145.0 ± 1.2	190.9 ± 3.6	331.4 ± 16.6	345.4 ± 13.4
Proposed	**109.9 ± 0.1**	**116.9 ± 0.8**	**144.9 ± 1.5**	**190.7 ± 3.6**	**317.0 ± 7.6**	**330.7 ± 7.9**
{20 Fog VMs U 10 VMs Cloud}						
FF	303.5	410.2	570.8	665.2	534.6	515.8
BF	159.7	523.15	328.9	381.1	334.8	414.0
GA	**109.3 ± 0**	111.2 ± 0.6	112.8 ± 0.5	113.3 ± 0.4	126.6 ± 0.9	167.7 ± 1.2
BLA	**109.3 ± 0.0**	**110.8 ± 0.0**	111.7 ± 0.1	112.4 ± 0.2	**125.5 ± 1.8**	**165.9 ± 1.4**
Proposed	**109.3 ± 0.0**	**110.8 ± 0.0**	**111.6 ± 0.0**	**112.3 ± 0.4**	125.8 ± 0.3	173.2 ± 0.3
{30 Fog U 15 Cloud}						
FF	282.6	560.4	621.5	690.7	615.7	649.9
BF	186.7	554.1	313.0	293.1	340.0	424.9
GA	109.2 ± 0.1	110.3 ± 0.2	111.0 ± 0.2	111.4 ± 0.2	111.2 ± 0.5	117.7 ± 1.0
BLA	**109.1 ± 0.0**	110.3 ± 0.1	**110.8 ± 0.2**	110.8 ± 0.2	**110.4 ± 0.2**	115.7 ± 0.3
Proposed	**109.1 ± 0.0**	**110.2 ± 0.1**	111.4 ± 0.1	**110.7 ± 0.1**	**110.4 ± 0.2**	**113.7 ± 0.2**

Bold values indicate the best found results

the number of VMs decreases the makespan of all algorithms. Moreover, all stochastic algorithms (the GA, BLA, and the IGA-POP) achieved better makespan values than the deterministic algorithms (FF and BF). It is observed that the stochastic algorithms have very similar performance when a large number of computing nodes (scenario 3: 30 FN and 15 CN) is applied, while the proposed scheduling algorithm clearly outperforms all other algorithms in the case of limited resources (scenario 1: 10 FN and 5 CN). Additionally, we can see that the stochastic algorithms have very similar performance when handling small task datasets, while the proposed scheduling algorithm clearly outperforms all other algorithms in the case of large datasets. Consequently, the superiority of the proposed algorithm in terms of the makespan is clear in the case of limited resources and large task datasets. In addition, the reported standard deviation values reveal the stable performance of the proposed scheduling algorithm.

The different scheduling algorithms are evaluated in terms of the total execution cost. Table 9 shows the obtained results.

Table 9 reveals that stochastic algorithms outperform the deterministic algorithms with respect to the execution cost in all cases. For deterministic algorithms, the BF algorithm significantly outperforms the FF algorithm. On the other hand, all stochastic algorithms have very similar performance on small task datasets, while the proposed scheduling algorithm saves a significant amount of execution cost compared to the other stochastic algorithms on large task datasets. For example, executing 400 tasks under scenario 1 (10 FN and 5 CN), the total costs of the GA, the PLA, and the proposed algorithm are 2842.3 ± 26.7, 2821.6 ± 16.9, and

Table 9 Execution cost of different scheduling algorithms (in G$)

Method	Tasks					
	N = 50	N = 100	N = 150	N = 200	N = 300	N = 400
{10 Fog VMs U 5 Cloud VMs}						
FF	319.2	696.9	1012.4	1311.8	2201.7	3371.9
BF	304.9	693.1	1108.7	1366.9	**1967.7**	**2628.5**
GA	295.9 ± 1.7	634.1 ± 3.1	1008.1 ± 3.4	1306.8 ± 4.7	2004.7 ± 11.6	2842.3 ± 26.7
BLA	$\mathbf{294.1 \pm 0.3}$	624.1 ± 1.1	1002.6 ± 4.4	1300.2 ± 2.8	1992.1 ± 3.9	2821.6 ± 16.9
Proposed	$\mathbf{294.1 \pm 0.3}$	$\mathbf{619.1 \pm 1.8}$	$\mathbf{989.6 \pm 2.8}$	$\mathbf{1282.7 \pm 2.3}$	1968.3 ± 9.3	2780.1 ± 12.3
{20 Fog VMs U 10 Cloud VMs}						
FF	292.0	643.0	1007.0	1300.2	1823.9	2442.5
BF	**205.8**	523.2	823.6	1090.8	1763.9	2317.1
GA	238.9 ± 2.6	513.0 ± 1.0	830.9 ± 2.5	1106.0 ± 7.8	1688.9 ± 8.6	2271.6 ± 2.8
BLA	233.8 ± 0.2	499.7 ± 1.2	814.7 ± 2.3	1091.6 ± 8.7	1673.3 ± 3.4	2264.7 ± 8.4
Proposed	233.9 ± 0.0	$\mathbf{494.1 \pm 2.6}$	$\mathbf{791.1 \pm 1.3}$	$\mathbf{1049.6 \pm 1.4}$	$\mathbf{1626.3 \pm 3.4}$	$\mathbf{2221.6 \pm 5.7}$
{30F Fog VMs U 15 Cloud VMs}						
FF	319.2	666.0	988.4	1279.5	1776.2	2295.5
BF	296.5	554.1	838.7	1083.3	1718.6	2294.7
GA	255.9 ± 1.0	531.3 ± 3.4	811.8 ± 0.8	1054.8 ± 5.1	1628.2 ± 6.0	2202.6 ± 16.7
BLA	$\mathbf{253.7 \pm 0.0}$	520.6 ± 1.9	796.4 ± 0.8	1040.0 ± 1.4	1609.3 ± 3.1	2192.8 ± 6.6
Proposed	$\mathbf{253.7 \pm 0.0}$	$\mathbf{516.5 \pm 0.7}$	$\mathbf{781.1 \pm 0.7}$	$\mathbf{1007.9 \pm 0.7}$	$\mathbf{1554.2 \pm 3.1}$	$\mathbf{2122.4 \pm 0.5}$

Bold values indicate the best found results

2780.1 ± 12.3, respectively. In addition, the small standard deviation value associated with the proposed scheduling algorithm reflects the performance stability of the proposed algorithm compared to the other stochastic algorithms.

The failure rate reflects to what extent the scheduling algorithms meet the deadline times of the real-time tasks. It represents the number of failed tasks whose expected finish times are greater than their deadline times. Table 10 shows the failure rate of the different scheduling algorithms under different scenarios.

Table 10 demonstrates that increasing the number of tasks increases the failure rate of all scheduling algorithms under certain VM combinations. For example, the proposed scheduling algorithm has a 1.2 ± 0.7 failure rate when executing 50 tasks under scenario 1 (10 FN and 5 CN), while it has a 45.3 ± 0.8 failure rate when executing 400 tasks under the same scenario. Additionally, increasing the number of VMs decreases the failure rate of all algorithms. For example, the proposed scheduling algorithm has a 45.3 ± 0.8 failure rate when executing 400 tasks under scenario 1 (10 FN and 5 CN), while it has a 0.8 ± 0.3 failure rate when executing the same number of tasks under scenario 3 (30 FN and 15 CN). Moreover, all stochastic algorithms achieved better failure rates than the deterministic algorithms. For the deterministic algorithms, the BF algorithm is better than the FF algorithm in all cases. The stochastic scheduling algorithms have similar performance, where no one algorithm can be marked as the best algorithm in all cases.

Finally, another performance metric, called the average delay time, is used for real-time task scheduling. It represents the average delay of failed tasks. The delay time of

Table 10 Failure rate of different scheduling algorithms (%)

Algorithm	Tasks					
	N = 50	N = 100	N = 150	N = 200	N = 300	N = 400
{10 Fog VMs U 5 Cloud VMs}						
FF	36.0	71.0	60.7	59.0	59.7	68.0
BF	24.0	65.0	57.3	60.5	56.7	65.3
GA	1.6 ± 0.9	**1.0 ± 0.5**	8.4 ± 1.3	22.3 ± 0.6	**42.3 ± 1.7**	**45.2 ± 2.4**
BLA	1.2 ± 1.1	1.0 ± 0.9	**8.0 ± 0.5**	22.8 ± 1.8	42.2 ± 0.9	46.0 ± 1.9
Proposed	**1.2 ± 0.7**	1.6 ± 0.9	8.1 ± 1.6	**21.7 ± 1.1**	43.3 ± 1.2	45.3 ± 0.8
{20 Fog VMs U 10 Cloud VMs}						
FF	48.2	52.0	58.7	60.5	70.3	66.8
BF	14.0	27.0	45.3	48.0	53.0	58.8
GA	**0.0 ± 0.0**	**0.0 ± 0.0**	**0.9 ± 0.4**	0.5 ± 0	**6.2 ± 0.4**	**16.9 ± 0.4**
BLA	**0.0 ± 0.0**	**0.0 ± 0.0**	1.1 ± 0.4	**0.3 ± 0.3**	6.9 ± 0.4	17.2 ± 0.7
Proposed	**0.0 ± 0.0**	**0.0 ± 0.0**	1.1 ± 0.4	**0.3 ± 0.3**	6.9 ± 0.2	17.7 ± 0.2
{30 Fog VMs U 15 Cloud VMs}						
FF	42.0	68.0	75.3	71.0	64.3	66.5
BF	18.0	33.0	41.3	43.0	50.7	54.5
GA	**0.0 ± 0.0**	**0.0 ± 0.0**	**0.0 ± 0.0**	7.0 ± 0.3	**0.6 ± 0.2**	1.1 ± 0.3
BLA	**0.0 ± 0.0**	**0.0 ± 0.0**	**0.0 ± 0.0**	**1.0 ± 0.0**	**0.6 ± 0.2**	**0.8 ± 0.3**
Proposed	**0.0 ± 0.0**	**0.0 ± 0.0**	**0.0 ± 0.0**	**1.0 ± 0.0**	0.7 ± 0.0	**0.8 ± 0.3**

Bold values indicate the best found results

a failed task indicates the difference between the expected finish time and the deadline time of the task. Hence, the average delay time can be computed by Eq. (21).

$$AverageDelayTime = \frac{\sum_{j=1}^{M} \sum_{i=1}^{NF_j} (EFT_{ij} - DRT_i)}{\sum_{j=1}^{M} NF_j} \tag{21}$$

where NF_j denotes the number of failed tasks on VM_j and the denominator denotes the total number of failed tasks. Table 11 shows the average delay time of the different scheduling algorithms.

Table 11 shows that the stochastic algorithms have a better average delay time than the deterministic algorithms in all cases. Additionally, increasing the number of tasks increases the average delay time of all algorithms in all cases. For example, the proposed scheduling algorithm has an average delay time of 0.3 ± 0.1 when executing 50 tasks under scenario 1 (10 FN and 5 CN), while it has an average delay time of 77.3 ± 3.5 when executing 400 tasks under the same scenario. Moreover, increasing the number of VMs reduces the average delay time of all stochastic algorithms. For example, the proposed scheduling algorithm has a 77.3 ± 3.5 average delay time for executing 400 tasks under scenario 1 (10 FN and 5 CN), while it has an average delay time of 2.6 ± 0.9 when executing the same number of tasks under scenario 3 (30 FN and 15 CN). However, the proposed scheduling algorithm has the best

Table 11 Average delay time of different scheduling algorithms (in s)

Method	Tasks					
	N = 50	N = 100	N = 150	N = 200	N = 300	N = 400
{10 Fog VMs U 5 Cloud VMs}						
FF	47.4	66.0	77.2	80.4	88.1	113.6
BF	22.5	68.0	91.7	82.9	93.8	94.9
GA	0.6 ± 0.6	1.1 ± 0.8	12.0 ± 1.0	25.8 ± 2.2	66.0 ± 5.3	80.1 ± 3.0
BLA	0.3 ± 0.2	1.4 ± 1.3	10.5 ± 0.9	24.9 ± 1.8	62.3 ± 4.7	76.7 ± 1.6
Proposed	$\mathbf{0.3 \pm 0.1}$	$\mathbf{1.3 \pm 1.1}$	$\mathbf{9.6 \pm 0.7}$	$\mathbf{24.3 \pm 0.9}$	$\mathbf{55.7 \pm 3.4}$	$\mathbf{77.3 \pm 3.5}$
{20 Fog VMs U 10 Cloud VMs}						
FF	53.3	68.9	134.8	159.0	128.1	133.7
BF	15.1	60.0	70.3	74.2	74.0	82.6
GA	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	1.1 ± 0.6	1.0 ± 0.5	8.8 ± 0.9	15.9 ± 1.0
BLA	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	$\mathbf{1.0 \pm 0.4}$	$\mathbf{0.4 \pm 0.4}$	7.9 ± 0.4	$\mathbf{15.2 \pm 1.3}$
Proposed	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	$\mathbf{1.0 \pm 0.4}$	$\mathbf{0.4 \pm 0.4}$	$\mathbf{7.8 \pm 0.4}$	17.4 ± 0.7
{30 Fog VMs U 15 Cloud VMs}						
FF	66.0	137.3	157.9	160.1	126.3	129.1
BF	22.7	37.7	52.6	58.7	61.3	78.2
GA	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	3.1 ± 0.8	3.8 ± 1.1	2.7 ± 1.3
BLA	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	$\mathbf{2.0 \pm 0.0}$	4.2 ± 0.9	$\mathbf{2.6 \pm 0.8}$
Proposed	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	$\mathbf{2.0 \pm 0.0}$	$\mathbf{3.1 \pm 0.1}$	2.6 ± 0.9

Bold values indicate the best found results

average delay time on all datasets with fewer than 400 tasks. Moreover, it obtains a comparative performance on the dataset with 400 tasks.

5.2.6 Elapsed Run Time Analysis

Furthermore, this experiment investigates the elapsed run time of the stochastic algorithms used. For fair comparisons, the deterministic algorithms are excluded from this experiment. Fifteen VMs (10 fog VMs and 5 cloud VMs) are used to execute several task datasets with different sizes. The obtained results are shown in Fig. 7.

Based on Fig. 7, it is observed that increasing the number of tasks increases the elapsed run times of all algorithms. However, the GA has the least elapsed run time, followed by the BLA and the proposed algorithm. Although the proposed algorithm comes last in terms of the elapsed run time, it offers many benefits compared to the other scheduling algorithms. It achieves a better performance in terms of the makespan, total execution cost, failure rate, and average delay time. Hence, the additional time taken by the proposed algorithm to efficiently explore and exploit the search space resulted in producing better schedules compared to the other scheduling algorithm, which positively reflected on the other performance metrics.

6 Conclusion

IoT technology plays a significant role in many aspects of modern life by providing a wide variety of applications, including home healthcare, smart manufacturing, smart agriculture, smart retailing, and smart transportation. Many of these applications require fast response times and low latency, which cannot be achieved by overwhelmed cloud resources. Fog resources are used together with cloud resources to achieve the requirements of real-time IoT applications. In this study, a semidynamic real-time task

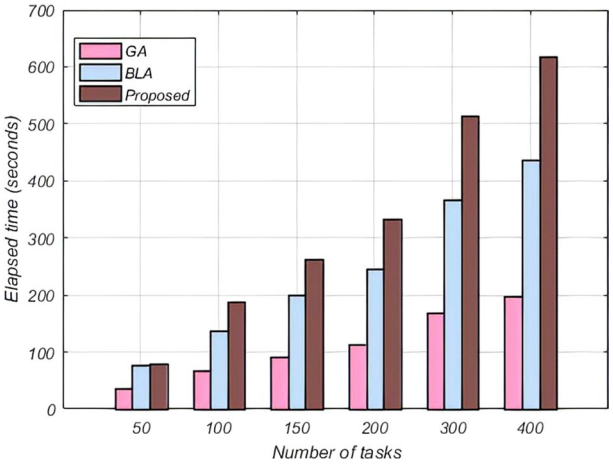


Fig. 7 The elapsed run time of the different stochastic scheduling algorithms

scheduling algorithm is proposed to schedule real-time tasks of IoT applications in the cloud–fog environment. The proposed scheduling algorithm approaches task scheduling as a permutation-based optimization problem. Hence, it employs the IGA-POP to provide the best permutation for the submitted real-time tasks. Then, the real-time tasks are assigned to the VMs that achieve the minimum expected finish time. Intensive experiments have been conducted to evaluate the performance of the proposed scheduling algorithm on datasets of different scales under different scenarios against FF, BF, the traditional GA, and the BLA. The obtained results reveal that the proposed algorithm can achieve a better balance between the makespan and the total execution cost than the other algorithms. Additionally, the conducted experiments show that the proposed scheduling algorithm has the minimum average delay time and failure rate. Additionally, it was noticed that the optimization algorithms (i.e., the proposed algorithm, the GA, and the BLA) have better performance compared to the state-of-art scheduling algorithms (i.e., FF and BF) with respect to the different performance metrics except the elapsed run time.

For future work, the proposed scheduling algorithm can be extended to be applicable in dynamic environments which use containers rather than VMs. The capability of containers environment is more suitable to the dynamic environment due to its flexibility and versatility to improve resource utilization. Moreover, the absence of virtualization layer in the container, incurs less performance overhead on the applications. In addition, the proposed can be adapted to address the workflow scheduling problem. Moreover, the proposed scheduling algorithm can be integrated with load balancing techniques. Additionally, other optimization algorithms, such as the shark optimization algorithm and whale optimization algorithm, can be evaluated for task scheduling in a cloud–fog computing environment. Finally, deep learning techniques can be employed to tackle the dynamic scheduling problem.

Funding Open access funding provided by The Science, Technology & Innovation Funding Authority (STDF) in cooperation with The Egyptian Knowledge Bank (EKB).

References

1. Čolaković, A., Hadžialić, M.: Internet of Things (IoT): a review of enabling technologies, challenges, and open research issues. Comput. Netw. **144**, 17–39 (2018). https://doi.org/10.1016/j.comnet.2018.07.017

2. Nguyen, B.M., Thi Thanh Binh, H., Do Son, B.: Evolutionary algorithms to optimize task scheduling problem for the IoT based bag-of-tasks application in cloud–fog computing environment. Appl Sci **9**(9), 1730 (2019). https://doi.org/10.3390/app9091730
3. Abohamama, A.S., Alrahmawy, M.F., Elsoud, M.A.: Improving the dependability of cloud environment for hosting real time applications. Ain Shams Eng. J. **9**(4), 3335–3346 (2018). https://doi.org/10.1016/j.asej.2017.11.006
4. Pham, X.Q., Man, N.D., Tri, N.D.T., Thai, N.Q., Huh, E.N.: A cost-and performance-effective approach for task scheduling based on collaboration between cloud and fog computing. Int. J. Distrib. Sens. Netw. (2017). https://doi.org/10.1177/1550147717742073
5. Li, G., Liu, Y., Wu, J., Lin, D., Zhao, S.: Methods of resource scheduling based on optimized fuzzy clustering in fog computing. Sens (2019). https://doi.org/10.3390/s19092122
6. Mukherjee, M., Shu, L., Wang, D.: Survey of fog computing: Fundamental, network applications, and research challenges. IEEE Commun. Surv. Tutor **20**(3), 1826–1857 (2018). https://doi.org/10.1109/COMST.2018.2814571
7. Liu, L., Qi, D., Zhou, N., Wu, Y.: A task scheduling algorithm based on classification mining in fog computing environment. Wirel. Commun. Mob. Comput. (2018). https://doi.org/10.1155/2018/2102348
8. Mahmoud, M.M., Rodrigues, J.J., Saleem, K., Al-Muhtadi, J., Kumar, N., Korotaev, V.: Towards energy-aware fog-enabled cloud of things for healthcare. Comput. Electr. Eng. **67**, 58–69 (2018). https://doi.org/10.1016/j.compeleceng.2018.02.047
9. Binh, H.T.T., Anh, T.T., Son, D.B., Duc, P.A., Nguyen, B.M.: An evolutionary algorithm for solving task scheduling problem in cloud-fog computing environment. In: Proc of the Ninth Int Symp on Inf and Commun Technol (pp. 397–404), Danang City (2018). https://doi.org/10.1145/3287921
10. Mishra, S.K., Puthal, D., Rodrigues, J.J., Sahoo, B., Dutkiewicz, E.: Sustainable service allocation using a metaheuristic technique in a fog server for industrial applications. IEEE Trans. Ind. Inform. **14**(10), 4497–4506 (2018). https://doi.org/10.1109/TII.2018.2791619
11. Ghobaei-Arani, M., Souri, A., Safara, F., Norouzi, M.: An efficient task scheduling approach using moth-flame optimization algorithm for cyber-physical system applications in fog computing. Trans. Emerg. Telecommun. Technol. (2020). https://doi.org/10.1002/ett.3770
12. Abohamama, A.S., Hamouda, E.: A hybrid energy: aware virtual machine placement algorithm for cloud environments. Expert Syst. Appl. (2020). https://doi.org/10.1016/j.eswa.2020.113306
13. Kimpan, W., Kruekaew, B.: Heuristic task scheduling with artificial bee colony algorithm for virtual machines. In: Joint 8th Int Conf on Soft Comput and Intell Syst (SCIS) and 17th Int Symp on Adv Intell Syst (pp. 281–286), Hokkaido (2016). https://doi.org/10.1109/SCIS-ISIS.2016.0067
14. Abdullahi, M., Ngadi, M.A.: Symbiotic Organism Search optimization based task scheduling in cloud computing environment. Fut. Gener. Comput. Syst. **56**, 640–650 (2016). https://doi.org/10.1016/j.future.2015.08.006
15. Mishra, S.K., Sahoo, B., Manikyam, P.S.: Adaptive scheduling of cloud tasks using ant colony optimization. In: Proc of the 3rd Int Conf on Commun and Inf Process (pp. 202–208) (2017). https://doi.org/10.1145/3162957.3163032
16. Gill, S.S., Buyya, R., Chana, I., Singh, M., Abraham, A.: BULLET: particle swarm optimization based scheduling technique for provisioned cloud resources. J. Netw. Syst. Manag. **26**(2), 361–400 (2018). https://doi.org/10.1007/s10922-017-9419-y
17. Reddy, G.N., Kumar, S.P.: Modified ant colony optimization algorithm for task scheduling in cloud computing systems. In: Smart Intell Computing and Appl (pp. 357–365) (2019). Springer. https://doi.org/10.1007/978-981-13-1921-1_36
18. Natesan, G., Chokkalingam, A.: Task scheduling in heterogeneous cloud environment using mean grey wolf optimization algorithm. ICT Express **5**(2), 110–114 (2019). https://doi.org/10.1016/j.icte.2018.07.002
19. Sangaiah, A.K., Hosseinabadi, A.A.R., Shareh, M.B., Bozorgi Rad, S.Y., Zolfagharian, A., Chilamkurti, N.: IoT resource allocation and optimization based on heuristic algorithm. Sensors **20**(2), 539 (2020). https://doi.org/10.3390/s20020539
20. Bitam, S., Zeadally, S., Mellouk, A.: Fog computing job scheduling optimization based on bees swarm. Enterp. Inf. Syst. **12**(4), 373–397 (2018). https://doi.org/10.1080/17517575.2017.1304579
21. Wan, J., Chen, B., Wang, S., Xia, M., Li, D., Liu, C.: Fog computing for energy-aware load balancing and scheduling in smart factory. IEEE Trans. Ind. Inform. **14**(10), 4548–4556 (2018). https://doi.org/10.1109/TII.2018.2818932

22. Yang, Y., Wang, K., Zhang, G., Chen, X., Luo, X., Zhou, M.T.: MEETS: maximal energy efficient task scheduling in homogeneous fog networks. IEEE Internet of Things J. **5**(5), 4076–4087 (2018). https://doi.org/10.1109/JIOT.2018.2846644
23. Rahbari, D., Nickray, M.: Low-latency and energy-efficient scheduling in fog-based IoT applications. Turk. J. Electr. Eng. Comp. Sci. **27**(2), 1406–1427 (2019). https://doi.org/10.3906/elk-1810-47
24. Chen, X., Xu, H., Huang, L.: Online scheduling strategy to minimize penalty of tardiness for real-time tasks in mobile edge computing systems. In: Int Conf. on Big Data and Comput (pp. 107–114), Guangzhou (2019). https://doi.org/10.1145/3335484.3335537
25. Deng, R., Lu, R., Lai, C., Luan, T.H., Liang, H.: Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption. IEEE Internet of Things J. **3**(6), 1171–1181 (2016). https://doi.org/10.1109/JIOT.2016.2565516
26. Kamal, M.B., Javaid, N., Naqvi, S.A.A., Butt, H., Saif, T., Kamal, M.D.: Heuristic min-conflicts optimizing technique for load balancing on fog computing. In: Int Conf. on Intell Netw and Collab Syst (pp. 207–219), Bratislava, Slovakia (2018). https://doi.org/10.1007/978-3-319-98557-2_19
27. Boveiri, H.R., Khayami, R., Elhoseny, M., Gunasekaran, M.: An efficient Swarm-Intelligence approach for task scheduling in cloud-based internet of things applications. J. Ambient Intell. Humaniz. Comput. **10**(9), 3469–3479 (2019). https://doi.org/10.1007/s12652-018-1071-1
28. Abdelmoneem, R.M., Benslimane, A., Shaaban, E.: Mobility-aware task scheduling in cloud-fog IoT-based healthcare architectures. Comput. Netw. **179**(9), 107–348 (2020). https://doi.org/10.1016/j.comnet.2020.107348
29. Fellir, F., El Attar, A., Nafil, K., Chung, L.: A multi-Agent based model for task scheduling in cloud-fog computing platform. In: 2020 IEEE Int Conf. on Informatics, IoT, and Enabling Technol (ICIoT) (pp. 377–382), Doha, Qatar (2020). https://doi.org/10.1109/ICIoT48696.2020.9089625
30. Nikoui, T.S., Balador, A., Rahmani, A.M., Bakhshi, Z.: Cost-aware task scheduling in fog-cloud environment. In: Int Symp on Real-Time and Emb Syst and Technol (RTEST) (pp. 1–8), Tehran, Iran (2020). https://doi.org/10.1109/RTEST49666.2020.9140118
31. Baniata, H., Anaqreh, A., Kertesz, A.: PF-BTS: a privacy-aware fog-enhanced blockchain-assisted task scheduling. Inf. Process. Manage. (2021). https://doi.org/10.1016/j.ipm.2020.102393
32. Singh, P., Singh, R.: Energy-efficient delay-aware task offloading in fog-cloud computing system for IoT sensor applications. J. Netw. Syst. Manag. **30**(1), 1–25 (2022). https://doi.org/10.1007/s10922-021-09622-8
33. Yadav, A.K., Garg, M.L.: Docker containers versus virtual machine-based virtualization. In: Emerging Technologies in Data Mining and Information Security (pp. 141–150). Springer, Singapore (2019)
34. Li, K., Peng, Z., Cui, D., Li, Q.: SLA-DQTS: SLA constrained adaptive online task scheduling based on DDQN in cloud computing. Appl. Sci. **11**(20), 9360 (2021). https://doi.org/10.3390/app11209360
35. Hoseiny, F., Azizi, S., Shojafar, M., Tafazolli, R.: Joint QoS-aware and cost-efficient task scheduling for fog-cloud resources in a volunteer computing system. ACM Trans. Internet Technol. **21**(4), 1–24 (2021). https://doi.org/10.1145/3418501
36. Yarpiz: Bees Algorithm (BeA) in MATLAB (2020). https://www.mathworks.com/matlabcentral/fileexchange/52967-bees-algorithm-bea-in-matlab. Accessed 27 June 2020

A. S. Abohamama was born in Egypt in 1985. He received B.Sc., M.Sc., and PhD. in Computer Sciences from Mansoura University in 2006, 2012, 2018, respectively. He works as an Assistant Professor at the Department of Computer Sciences, University of Mansoura. His current research interests include Distributed Systems, Cloud Computing, IoT and Big data analytics, Image Processing, and Biometrics.

Amir El-Ghamry is currently a Lecturer in the Computer Science Department at Mansoura University. He had been a Visiting Scholar for Ph.D. degree in the Department of Computer Science at the University of Texas at Dallas from 2016 to 2018. He received his Ph.D., M.S., and B.Sc degrees in Computer Science from Mansoura University in 2019, 2012 and 2006, respectively. He had been a Member in Open Event Data Alliance (OEDA) Research Project at the University of Texas at Dallas from 2017 to 2018. He is a Project Consultant for (Minerva – Afghanistan) Project at the University of Arizona, from 2019 until present. His research Applies and Extends Machine Learning, Security, Optimization, IoT Middleware, Big Data Analytics and Natural Language Processing.

Eslam Hamouda received his B.Sc. degree in 2006, M.Sc. degree in 2011, and Ph.D degree in 2016 from Mansoura University, Egypt. He was a Visiting Scholar in the Department of Computer Science and Engineering, University of North Texas, Denton, TX, USA. Currently, he is an Associate Professor in Computer Science Department, Faculty of Computers and Information, Mansoura University, Egypt, he also works as an Assistant Professor in Computer Science Department, Faculty of Computers and Information, Jouf University, KSA. His research interests are in the areas of Machine Learning, Biometrics, Optimization, and Pattern Recognition.

Authors and Affiliations

A. S. Abohamama[1] · Amir El-Ghamry[1] · Eslam Hamouda[1,2]

✉ A. S. Abohamama
abohamama@mans.edu.eg

Amir El-Ghamry
amir_nabil@mans.edu.eg

Eslam Hamouda
eslam_foad@mans.edu.eg

[1] Faculty of Computers and Information Sciences, University of Mansoura, 60 El-Gomhoreya Street, Mansoura 35516, Egypt

[2] Faculty of Computers and Information Sciences, Jouf University, Jouf 2014, Saudi Arabia