Received November 23, 2021, accepted December 6, 2021, date of publication December 13, 2021,
date of current version December 28, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3134941

Multi-Objective Accelerated Particle Swarm Optimization With Dynamic Programing Technique for Resource Allocation in Mobile Edge Computing

TAHA ALFAKIH[1], MOHAMMAD MEHEDI HASSAN[1], (Senior Member, IEEE), AND MUNA AL-RAZGAN[2]
[1]Department of Information Systems, College of Computer and Information Sciences, King Saud University, Riyadh 11543, Saudi Arabia
[2]Department of Software Engineering, College of Computer and Information Sciences, King Saud University, Riyadh 11543, Saudi Arabia

Corresponding author: Mohammad Mehedi Hassan (mmhassan@ksu.edu.sa)

This work was supported by King Saud University, Riyadh, Saudi Arabia, through the Researchers Supporting Project, under Grant RSP-2021/206.

ABSTRACT Mobile edge computing (MEC) is a powerful new technology with the potential to transform and decentralize the way our cell phone networks currently work. The purpose of MEC is to process the intensive mobile applications in the available resources, which are embedded in the base station of the cell phone systems and closer to users (i.e., MEC support stations). We assumed that the telecommunications base station supports MEC, which provides edge computing with tiny latency. However, the problem of inevitable optimization emerges in terms of the quality of service (QoS) and user experience (QoE). Therefore, MEC services provide integrated services close to end-users to achieve QoS and QoE. This study examined how to jointly optimize resource allocation when offloading tasks from mobile devices (MD) to edge servers (ES) in MEC systems, thereby minimizing the computing time and service cost. The study's main insight is that offloaded tasks can be delivered in a scheduled manner to the virtual machines (VMs) in the ES to minimize computing time, service cost, waste over the capability of the ES, and maximum associativity ($\mathcal{A}_{E,X}$) of a task with an ES to maintain MD mobility. We present a dynamic task scheduling and load-balancing technique based on an integrated accelerated particle swarm optimization (APSO) algorithm with dynamic programming as a multi-objective. The proposed method was compared with the standard PSO, APSO, and PSO-GA algorithms using experimental simulations. The results show that the proposed method outperformed these algorithms, with a reduction in task makespan of 30% and an increase in resource utilization of 29% observed compared to PSO-GA. Additionally, the proposed method was associated with reducing service cost and waiting time compared to the other algorithms and improvements in the fitness function value.

INDEX TERMS Service-oriented computing (SOC), web services composition (WSC), web service (WS), web service selection (WSS), ant colony system (ACS).

I. INTRODUCTION
In recent years, there has been explosive growth in computationally intensive mobile applications, including augmented reality, image and signal processing, and online gaming. This has led to substantial computational demands on resource-limited mobile devices (MD). MDs are limited in computation, storage capacity, and battery, which means there is a growing demand to offload intensive tasks to powerful remote computing platforms as a service. Mobile cloud computing (MCC) introduces powerful computing as a computation offloading model for MDs [1]. It provides a pool of resources with CPUs, high power, and storage capabilities as cloud servers to compute specific tasks. However, the considerable distances between cloud servers and MDs lead to significant communication costs in response time

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

delay, which negatively impacts real-time applications [2]. Therefore, recently, the remote cloud's capabilities in terms of computation and storage have been partially offloaded to edge servers (ES), which are positioned closer than cloud servers are to the MDs. The emerging mobile edge computing (MEC) paradigm provides numerous services and powerful remote computing for both enterprisers and users [3].

In MEC, user devices can exploit cloud computing capabilities at the mobile network edge. MEC is implemented through the intensive deployment of ES on the cellular network edge and the base station (BS), all of which have the necessary resources in terms of storage and computation. The main objectives of MEC are to ensure service deployment and effective network operation, offer an improved QoE, and decrease latency [3], [4]. Noteworthily, in the smart cities that can benefit from offloading operations to ES, this process gives rise to cyber-physical social systems (CPSS). These systems utilize MEC for various purposes, including geological surveys, drone delivery services, and traffic violation tracking cameras, among other applications.

An ES executes an offloaded task itself rather than sending it to a remote cloud. Therefore, MEC can meet QoS requirements and enhance the quality of the QoE, which it can achieve – for example – through low response times and minimal energy consumption. Despite the massive potential of MEC, there are still challenges to overcome. As stated before, real-time mobile applications are highly sensitive in terms of response time and energy consumption. For this reason, the most prominent challenges facing MEC are long execution times, scarcity of resources, high cost, mobility, and optimization of resource allocation [5].

The main idea in MEC is how to manage resources within the edge node while satisfying critical requirements in terms of time processing, cost of service, and mobility. We recommend that the offloaded task is processed in the same edge in which it was initially mapped, irrespective of the fact that the user may have left the edge node's coverage area. We suggest that the task is not migrated to another node (i.e., one closer to the user) because this increases the processing time; instead, the result should be transferred only through the nearest edge node. Optimal allocation is a critical issue because overuse of resources causes scarcity, whereas underutilization leads to drastically lower QoS in the system. Additionally, the optimized allocation of tasks to virtual machines (VM) reduces latency and increases system throughput by trading off the load across all available VMs.

With the above considerations in mind, it is necessary to develop a system for optimizing resource allocation when offloading tasks to minimize computing time ($\boldsymbol{T_e}$) And service cost and maintain mobility, where the tasks can be processed locally and offloaded to the ES. Furthermore, efficient resource allocation is required concerning all of the conflicting computation-offloading objectives, and the waste over the capability of the ES should be reduced. In addition, due to the limited resources of ES, resource utilization should be improved so that QoS requirements are met (e.g., response time requirement). Moreover, the heterogeneity of edge node resources, user mobility, and the physical distribution of MDs present additional challenges for offloading computation in ES. Most studies have highlighted these challenges and proposed solutions to resolve them [6], [7]. However, prior studies have been restricted to optimizing QoS through dynamic service, service placement, and network selection issues; they have overlooked the impact of distributing tasks inside ESs as resources management and load balancing. Furthermore, most studies have not applied efficient multi-objective optimization to ensure the maximum utilization of limited edge node resources and reduce the computing cost. This study sought to solve these challenges and improve resource management by proposing the MOAPSO-DP algorithm.

The main challenges addressed in this study are the following:

1) How can $\boldsymbol{Minimize}(\boldsymbol{T_e})$ be achieved? The tasks are placed in a wait queue to enable each task to be processed separately, which leads to an increase in the service time $\boldsymbol{T_e}$ and a delay in terms of responsiveness.
2) How can the maximum associativity of a task ($\mathcal{A}_{E,X}$) be obtained with an ES? This challenge concerns how to maintain the computation services when the MD moves mobility) from the coverage area of the ES into another coverage area. Therefore, it is necessary to measure the time that the MD has been connected to the ES.
3) What is the best way to reduce the number of VMs (i.e., $\boldsymbol{Minimize}(\boldsymbol{\eta_j})$), thereby minimizing the service cost? Regarding this challenge, when creating VMs, $V = \{v_1, v_2, v_3, \ldots, v_n\}$ overcomes this challenge subtask assigns to an individual VM to parallel processing to reduce the processing time. Therefore, this leads to waste over the capability of the ES and, as a result, an increase in the service cost ($\boldsymbol{C_i}$). Furthermore, the tasks are stacked in the queue to wait for available VMs to receive new tasks, which exacerbates the delay in computation.

An elegant way to overcome these three challenges is to regard them as conflicting objectives. Conceptualized in this way, the problem becomes one of achieving efficient resource allocation with respect to the conflicting objectives of the computation-offloading task. In the literature, problem45s in this class are known as multi-objective optimization (MOO) problems. MOO problems require specific model designs to handle multiple objectives and identify the best possible operating point. In this study, we formulate these issues and find the optimal solutions for MOO. MOO is an essential aspect of optimization activities because most optimization problems in the real world involve conflicting objectives. Therefore, multi-objective evolutionary algorithms have been used to develop efficient solutions to problems involving multiple conflicting objectives; an example is the accelerated particle swarm optimization (APSO) algorithm [8]. APSO is an evolutionary and heuristic algorithm that has been used in diverse models to solve challenges such as task scheduling in cloud computing.

This study presents a scheduling model for offloading tasks in an ES, the wider context being to satisfy the multi-objectives of improving performance in the following variables: task processing time, mobility, and VM processing cost. To address these multi- objectives, we need to deal with these objectives as one objective with three dimensions based on the MOAPSO-DP model. A notable feature of the model is that it assigns equal priority to the three conflicting objectives, and it also designs problems using a mathematical model. We also evaluate these objectives separately based on a weight value policy to elect the best solution. The MOAPSO-DP algorithm is used to select the optimal VMs depending on the three conflicting objectives. To see how these objectives conflict with each other, consider how reducing the task processing time (TPT) leads to an increase in the VM cost. In addition, to maximize task associativity ($\mathcal{A}_X$), we should increase the number VMs, which also increases the service cost. The MOAPSO-DP model evaluates each objective separately using the integration between the APSO and dynamic programming strategy, solving the knapsack problem to find the appropriate VM for each offloaded task.

The main contributions of this study are the following:

1) Resource allocation management in an ES context, most notably using an efficient method to improve resource utilization,
2) Reduction in the number of VMs by ensuring that VMs are created on-demand according to the number of offloaded tasks and a load balancing between them to eliminate the problem of wasting VM resources. This contribution was achieved using dynamic programming to solve the knapsack problem, resulting in a substantial reduction in the processing cost.
3) Mapping between tasks and VMs to achieve the optimal position increases system throughput by balancing tradeoff load across all available VMs.
4) Improvement in the offloaded task's associativity with VM to support mobility by making scheduling non-preemptive. This contribution was achieved using the APSO.

The rest of the paper is organized as follows: Section 2 presents the related work review. In Section 3, we outline the architecture of MEC. In Section 4 system model, and presented the problem formulation in Section 5. Then in Section 6, we describe our proposed model MOAPSO-DP. Section 7 implements the model on a simulation and provides the analysis of the results, followed by the conclusion in Section 8.

II. RELATED WORK

A rich body of literature exists on resource allocation methods in general cloud computing and mobile edge computing (MEC) in particular. However, sufficient attention has not been paid toward task scheduling in the context of multiple optimization objectives in complex applications. Instead, at present, the resource allocation process occurs after the decision-making process is applied to offload computation to the MEC (resulting in either partial or full offloading), which highlights the need for a proper allocation of resources. If it is impossible to offload, then the partitioned and paralleled application is only allocated on a single edge node for the execution. In contrast, the partitioned application is distributed over several nodes.

The total size of the tasks offloaded to the MEC system should be compatible with processing time requirements and energy consumption threshold [9]. The application's priorities are to determine where tasks should be allocated based on the availability of computing resources at the MEC system. The authors in [10] assumed that there are several nodes in the density area by the user equipment (UE), which enabled MDs to access the MEC system using the improved node B (eNB). Furthermore, an offered efficient method and policy by propose an equivalent discrete-time Markov decision problem (MDP) framework. Noteworthily, the outcome of this method was high computational complexity. The authors overcame the problem by developing an index policy of the application assignment, which is calculated for each eNB based on its available computing resources. The eNBs are broadcast to this index policy, and the MDs select a suitable MEC server. The results demonstrated that computation delay and energy consumption are minimized. The research in [11] is similar to the studies of [9] and [10], where the main goal in each article was to minimize latency, energy consumption, channel overload, execution resource overload, and VM migration cost. In [11], the authors used enhanced small cells (sCeNB) as a server node at the MEC system, and each of the MDs mapped the VM at the SCeNB. This minimized the communication latency because the high-quality transmission of data characterizes the SCeNBs.

The advantage of the methods mentioned above is that they reduce computation delay, energy consumption, communication overhead, computing overhead, and migration cost. Reducing computation time is achieved by using a multi-resource that allocates the SCeNBs as a cluster, thereby avoiding sending to remote cloud computing [12], [13]. The proposed cooperative game approach creates a nearby sCeNB in the same cluster based on the distance between them and the overlap of their service coverage; this reduces the execution delay through the parallel offloading of application portions and their distribution to VMs at the sCeNB in the same cluster. Data compression is applied to compress the offloaded data before transmission to reduce the data size. The problem of jointly optimizing computation offloading, data compression, and resource allocation to minimize energy consumption under the latency constraint and finite MEC computation capacity is considered. In order to solve this non-convex problem, the authors convert it into a convex one and apply convex optimization [14].

The research [15] goals to minimize the latency. This is performed using a one-dimensional search model, which attempts to make an efficient offloading decision based on the following factors: state buffer of queuing applications, available energy in the ES, and the status of communication between the MD and ES. The advantage of this method is compared with the policies of greedy offloading, cloud computing, and local computing. The results of the simulation framework show that latency is reduced by up to 81%. The drawback of the model is that in order to make a decision, the MD requires feedback from the ES. In [16], the authors proposed a low-complexity Lyapunov optimization-based dynamic computation offloading (LODCO) algorithm, which decides either local computation or migration each time. When offloading to the MEC system, the proposed algorithm can reduce delays by up to 65%. As mentioned in the paper [15], one limitation of the methods is that the authors did not consider power consumption when user experience (UE) offloading decision.

In [17], the authors improved the performance capability to leverage edge servers to execute offloaded tasks and accommodate them. In the research, the authors created a pool of co-located devices as a local cloud service at the ESs and, in turn, enabled the MDs of multiple clients to be configured into coordinated computing services at the cloud, despite an increase in the participation of MDs. The authors in [18] introduced a new model of computation offloading in the MEC paradigm. The idea of the model was to support the use of virtual resources in the ES to transfer the burden of resources, save energy, and improve application performance. This model is known as mobile virtual resources (MVR), and it automatically moves from a paradigm of single device execution (on the client) to distributed execution (on the client and ES). The research in [19] presented a novel model for computation offloading from an MD to an ES with the availability of the highest CPU to minimize latency at both the MD and MEC system. The main concept of the proposed model was based on an estimated value of round-trip time (RTT) between the ES and the MD. It depended on the signal quality of the radio network information service (RNIS) and an application programming interface (API) to decide whether to offload or compute tasks.

An opportunistic computation offloading scheme was developed for the MECC model to implement data mining tasks in edge networks while minimizing latency and energy consumption [20]. The authors in [21] designed a distributed computation offloading algorithm that exploits a Nash equilibrium technique to achieve high performance in computation offloading and scales and increase the user size. This involved the use of distributed learning to solve server mode selection on the server-side [22]. In [6], the authors analyzed and evaluated the performance of computation offloading from MDs to the small cell cloud. The research in [23] focused on the offloading of deep neural network (DNN) based applications into ESs from the standpoint of resource-limited devices (e.g., MDs) that cannot support DNNs. However, the offloading process of these dense applications is time-consuming, which lengthens the computation time and impacts the user experience. In their research, the authors used a parallel scheduling process with genetic algorithms (GA) and the greedy algorithm to address the problem and reduce processing time.

In [24], a resource scheduling model was proposed as a two-level system in the fog nodes using the non-dominated sorting genetic algorithm (NSGA-II). The results demonstrated that the proposed scheme reduced service latency and improved the stability of task execution. The main idea of the NSGA-II is to build the specific chromosome population non-dominated sets, where non-dominated sets in several levels are sorted via the individual's crowding distance. The authors in [25] proposed a novel resource allocation model for computations involving big data. The idea is to select computing resources like cloudlets, considering the mobility, quality of AP signal, QoS, and workload distribution at the corresponding cloudlets. The assumption is made that there is a master cloud, which selects the appropriate cloudlet based on information from the MD. The authors considered the movement of the user with both change speed and track, and they used a model of SMOOTH random mobility to predict direction and velocity on the next time step.

In [26], the authors aimed to reduce energy consumption and resource provision by developing a hybrid metaheuristic algorithm known as genetic simulated annealing-based PSO. PSO is an evolutionary and heuristic algorithm that has been used in various models to solve diverse issues, including task scheduling problems in cloud computing. Non-stationary time series prediction for the IoT in the ES avoids accelerating convergence and an early convergence rate, which is different compared to standard PSO [27]. Minimize the computation time and consumption of energy by making decisions about computational resource allocation and computation offloading. The offloading decision determines the computing location for the computational application (e.g., mMEC offloading or sMEC offloading). However, resource allocation focuses on the problem of how to meet sMEC and mMEC resources for task execution [28]. The proposed module in the research [29] was based on an integration between PSO and GA to minimize data transmission and data storage for IoT applications. This module studies task offloading of the multi-service with multiple ESs and optimizing the MEC access network selection. The authors described MEC network selection as an NP-hard problem and proposed a PSO-based algorithm as a solution. The algorithm focused on service placement and network selection by mapping each task to the appropriate edge [30]. It consisted of a three-tier architecture containing a centralized cloud, roadside cloudlet, and vehicular cloud.

In the research of [31], the authors proposed a hybrid adaptive PSO (HAPSO) algorithm as an optimization process for resource allocation. The authors focused on three objectives (i.e., a MOO problem): namely, to enhance network latency, reduce total energy consumption, and increase availability [30]. They examined the migration strategy in the MEC to transfer services from the early nodes to other edge nodes that can offer services to meet QoS by resource allocation to reduce time service and energy consumption. In another study undertaken by [32], the authors developed a particle swarm-based service migration scheme integrated by the modified quantum particle swarm algorithm and queuing delay prediction algorithm (DCRA). Reference [31] argued that PSO is an ideal option for distributed load processing in independent edge clusters in the context of multi-cluster edge architectures for IoT management. The authors used PSO to decide which edge node to allocate a request to. The main idea was to divide the edge layer into smaller clusters, with each cluster having a central coordinating node; each node also communicates with other nodes by the controller [33]. In another research project, the authors integrated the greedy strategy PSO with dynamic PSO algorithms to reduce the cost and maximize the performance of the computation offloading process [34]. In the literature, other researchers have developed a location-based mapping scheme that uses the positions of particles and the current best solution to generate high-quality solutions [35]. The authors in [36] used the ant colony optimization (ACO) algorithm to solve MEC resource allocation and the computation offloading problem, focusing on selecting the MEC node in the shortest possible time.

Based on the extant literature, it is evident that scheduling management in the MEC context is an important issue. It influences edge cloud performance and directly impacts costs for edge cloud users and providers. At the same time, previous studies have primarily investigated providers' benefits concerning specific objectives, including improving techniques for resource utilization, reducing service cost, and increasing the throughput of the cloud computing system. Therefore, in the present research, we address these objectives from other perspectives using the MOO process. MOO has featured prominently in various related studies in recent years. Additionally, our model supports mobility, which promotes service continuity, and this factor has not been considered adequately in previous studies.

We conceptualize computing time as having a local execution component and a remote execution component. In remote execution, the time is present as three cases: communication delay to the ME, execution time at the ME and the result received from the ME. We also focus on the dynamic strategies, present an optimization model, and formulate the problem of task allocation scheduling. The load balancing algorithm manages edge resources, such as finding the most efficient VM to offload a task within the expected response time. It also determines each VM's capability in the ES, serves new tasks to the most efficient VM, and minimizes the response time. This study uses APSO with dynamic programing as a MOO approach to allocating incoming requests to the respective VMs. Significantly, the results reported in the later parts of this paper outperform many previous studies and, as such, are state of the art.

In the next section of this paper, an explanation is given of MEC architecture and system models. The problem formulation is presented, and the proposed method based on APSO is described in detail.

III. MOBILE EDGE COMPUTING ARCHITECTURE

MEC [4] is a new technique and paradigm that is currently being standardized in an ETSI Industry Specification Group of the same name and which offers computing resources installed over a radio access network (RAN) near MD. Therefore, these cloud centers are called mobile edge hosts. They are essentially computing equipment installed near the RAN or within or near the base station (BS). Hence, the MEC architecture is managed by the network operator. The computing resources within the mobile edge host must be virtualized; access to these virtualized resources takes place via access points (APs). Therefore, the goal of MEC is to minimize latency, ensure effective network operation and service transfer, and improve the user experience [4]. Various challenges are associated with MEC [37], including service synchronization and orchestration between the cloud servers and the ES and central cloud. Connectivity at the edge computing infrastructure may also be interrupted due to mobility, which has produced the problem of seamless service delivery. Also, service-centric structure (SCS) is marked by a situation, and the changes are centered on the service itself rather than its position; as a result, standard IP-based operation problems become infeasible, particularly regarding the handling of interactions between servers and clients and selecting the ES that the user is employing to offload their intensive task. Accordingly, resource management is the most prominent challenge in this technology because it reduces cost and serves the most significant possible number of devices [22].

The other challenges of edge cloud computing services may not permanently suppose the availability of local infrastructure and the allocation of computing resources at the ES. The MEC paradigm can be viewed as a natural extension of the evaluation of mobile BS. Therefore, as we progress into the new era of 5G technology, MEC appears as a new product of the evolution of BS from pure communication, which provides computational capacities integrated with the BS.

As shown in Figure 1, the MEC architecture is divided into the mobile edge, terminal, and remote cloud layers. The terminal layer typically consists of collected resource requestors, including cell terminal units such as cellular telephones and personal computers. The latency sensitive tasks and intense tasks created by these devices request resources from the mobile edge layer, which consists of computing servers, routers, gateways, and edge resource providers on the BSs. In turn, the intensive tasks migrate to the core layer (remote cloud and data centers). Computation offloading enables applications to benefit from remote computer resources; this is achieved using partitioning that minimizes response times by moving compute-intensive tasks to the remote cloud. However, memory offloading includes partitions of the application state across mobiles and nodes to mitigate memory constraints and reduce offloading overheads by computation. In contrast, network offloading minimizes network traffic by partitioning application states across MDs and nodes at edge locations in a mobile network.

Therefore, it transmits only the objects that are used at the device.

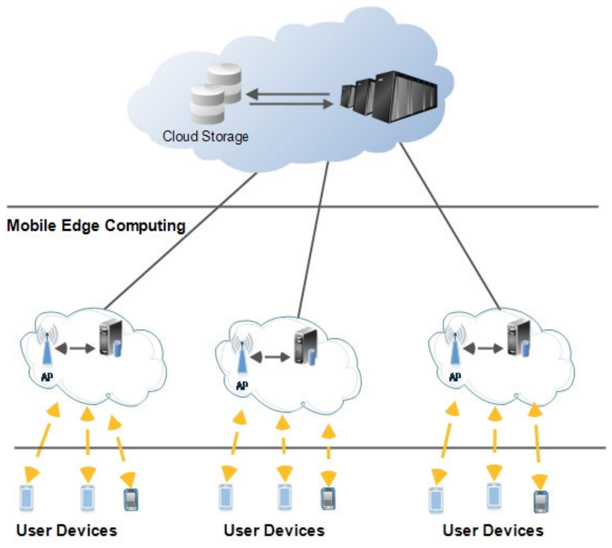


FIGURE 1. The architecture of mobile edge computing.

MDs have exploited the mobile cloud computing (MCC) infrastructure in smart cities to improve the QoS, minimize power consumption, and reduce latency by offloading/uploading services to powerful remote servers and other pooled MDs. Application partitioning is a required step for computation offloading, and it is considered critical in determining which parts can be offloaded and which cannot. Due to the influence on the computing process, the partitioning algorithms play an indispensable role in the performance of the offloading system.

Previous studies such as [38] used the Ford-Fulkerson method to resolve the partition computation offloading problem. The authors applied the maximum-flow minimum-cut (MFM) approach, representing the network as the application graph model and the program as a node. Also, it divides the network graph based on the label theory of propagation. According to the results of [37], the partitioning algorithm achieved the best performance in terms of computation offloading but was unsuitable for most applications when using 5G networks. From our perspective, the MOO method will be effective because there are multiple constraints for dynamic offloading. The MOO [39] introduced decision-making with various criteria concerned with mathematical optimization models involving multi-objective functions that must be optimized concurrently. The migration and balancing of the workload across a set of VMs depend on the availability of capability and capacity across all VMs. Migration will reduce the load on the current VM and increase the response time and execution speed [40]. The intensive components need to be offloaded to compute or store and schedule them. The migration module is required to make decisions regarding offloading, saving energy, and reducing latency by the remote cloud. The research [41] provides problem optimization by a mathematical analysis for joint wireless-aware resource allocation of the mobile application and cloud offloading.

Other remote cloud resources surround MDs, including fog computing systems, cloudlets, and peer devices. When an MD does not need to send a service directly to the cloud (due to cost or latency), it is possible to send it to an existing proximity edge network or cloudlet. If the MD cannot catch any available edge computing, then it will offload the service to the remote cloud; alternatively, in the worst case, it will execute the tasks locally. Therefore, a user receives a real-time response by one-hop, low-delay, low-cost access to the cloudlet, along with high bandwidth. As remarked in [42], "[this] is similar to a small data center located on designated areas and connected to a remote cloud server via the internet."

Security and privacy in the context of edge paradigms are critical, and many studies dealt with those issues that threaten the usage of resources in mobile edge computing [43], [44]. In this study, we did not address security issues; we will do detailed research in the future to address security issues related to the usage of the MEC resources.

IV. SYSTEM MODELS

We consider a MEC system with two sides: MDs and edge servers embedded at the telecommunications base station (BS). Also, we assume that the tasks have been offloaded and have already reached the edge server. Accordingly, this section describes the process in the edge computing model, including computation, offloaded tasks, task scheduling management, and edge resources management. In turn, we formulate a task scheduling problem for reducing computation latency and minimizing the service cost, which is caused due to wastage of the edge server (ES) capability.

Figure 1 illustrates how the monitoring and control unit creates elastically virtual machines (VMs) in the edge server to handle the offloaded tasks. The first VM is created to satisfy the processing requirements of the offloaded task. In turn, the VM control unit creates a mirror of the original VM to generate other VMs, thereby occupying the remaining capabilities of the ES with similar specifications as the first VM.

A. SERVER MODEL

We assume that an ES has limited resources but high computational power. We also assume that each ES can run several VMs instances, allowing them to compute offloaded tasks submitted by MDs. The resource allocation management algorithm runs on the ES at regular, scheduled intervals. The ES has four major units: the application clone unit, MARC unit, mobility, and communication unit, and scheduling and balancing unit.

The control edge cloud computing controller (CE3C) is responsible for the most complex computation: namely, finding the mapping between ESs that are close to each other, as well as the optimal mapping of MDs to the ESs, which is known as associativity. Additionally, the mapping of offloaded tasks to the MDs must be undertaken because each task contains $((X_{Id}, Data, D_{id})$, where $\boldsymbol{X_{Id}}$ is the task ID and D_{id} is the MD's ID, from which the application is sent. When the MD moves, the CE3C delivers the results to the MD through the closest ES; this is achieved by defining the location of the destination MD and connecting it to the nearest edge server.

The mobility and communication unit contains information about the historical mobility of each MD that is connected to the ES, as well as the geographic locations (Xe, Ye). Additionally, the mobility unit analyzes the patterns of device mobility to predict the speed and direction at future time steps based on the MD's historical information. The request scheduling and balancing unit receives all offloaded tasks and schedules them after a certain interval ε. Further challenges to solving the resource allocation problem thus arise from the varied task response times $\boldsymbol{t_p}$ of the task, as well as the movement of MDs.

Figure 2 shows a typical cross-server resource management scenario in the context of a MEC model. Multiple VMs with limited resources are created on the ES to serve MDs. Each device can independently send its tasks through ESs, while each ES that deploys the applications can process tasks concurrently. This involves creating multiple VMs to meet the task in terms of computation requirements, thereby achieving a balanced distribution of the workload among VMs.

We assume that the MD has offloaded a task $\boldsymbol{X}$ to the ES, after which the ES separates it into sub-tasks $\boldsymbol{X} = \{\chi_{\mathrm{i}}\}$. This is achieved using the solver unit to serve the tasks concurrently by accommodating it in the VMs without waste over the capability. Accordingly, the VMs are filled as mentioned above to process the sub-tasks on the VMs one by one (i.e., thereby reducing service cost and service time through concurrent processing). Therefore, we have $\mathrm{X}_{\upsilon} = \{x_{\upsilon,1}, x_{\upsilon,2}, \ldots, x_{v,j}\}$, where x_{υ} is the task indicator vector of υ^{th} VM and $x_{v,j}$ is the indicator of the j^{th} sub-task. If the VM (υ) serves the sub-tasks j, then $m_{v,j} = 1$; otherwise, $m_{v,j} = 0$. The task details for all VMs can be expressed using a task indicator matrix $\boldsymbol{X} = \{X_1, X_2, \ldots, X_{\mathrm{n}}\}$, where the υ^{th} a row is the task indicator vector of the ES υ. Accordingly, regarding the supply and demand relationship between the offloaded tasks and the VMs on the ES, each task bids on each VMs' resources to receive the services. The VMs with limited capacity simultaneously send requests for their resources.

B. EDGE RESOURCES MANAGEMENT MODEL

Resource scheduling is needed to minimize computation latency for tasks offloaded to the ES and ensure current resources can serve the largest number of tasks at the lowest cost. Resource allocation management control (RAMC) is concerned with assigning the tasks submitted from the resource scheduling unit into the appropriate VMs, as shown in Figure 1. The MARC is selected according to the CPU performance. It occupies the VM's remaining capabilities, which ensures that resources are not wasted, which is called

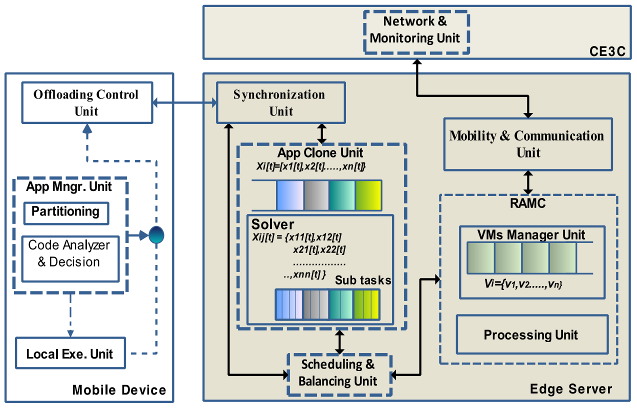


FIGURE 2. Typical resource management scenario of mobile edge computing model.

a waste over the capability of the ES. Each task uses an individual VM to undertake parallel processing to reduce processing time, but the result is a waste over the capability of the ES; this increases the service cost and stacks the tasks in the queue to wait for available VMs to receive new tasks, and it also exacerbates the computation delay. The MARC is also responsible for task migration between VMs. Whenever a task requests resources from the ES, the MARC communicates with another VM and allocates the appropriate VM resources to perform the tasks. This is known as resource scheduling among VMs.

Resource scheduling among VMs follows two principles: first, the availability and resource utilization of VMs, and second, the capability of the VM to serve the submitted task. Although resource maintaining leads to serving the largest number of tasks at the lowest cost, if wasted VM capability is exploited, the user will not bear the additional service cost; the service time will also decline due to the fact that processing occurs concurrently. Consequently, when scheduling resources among the VMs, it is also necessary to consider ES resource utilization.

C. CE3C MODEL

We assume that an intensive application on the MD requests offloading to the edge server (ES). This is the basis for designing an application in which one part involves local processing, and another part requires computation offloading to a close ES. When the tasks arrive at the ES, they are placed in a scheduled queue to start processing. Each task is divided into sub-tasks, and VMs are created within the ES to execute the tasks in parallel. Due to MD mobility, we assume that after the task is offloaded to the ES, which is located in the ES's coverage region, the MD moves and exits the initial coverage area. The MD does not receive the task results from the current ES. Therefore, the main challenge is how to return the results to the MD and satisfy the task's time requirements.

$$T_{e.m} < T_{l.m}; \quad \forall e \in ES, \ x \in X$$

We suppose that there exists a CE3C that works by tracking the MD that offloaded the tasks. It can also predict the amount of time that the MD will remain in the ES's coverage area and the service on the ES, additionally predict the next direction of the MD will go to receive thereby enabling the MD to receive the output via the closest ES after entering its service range.

V. PROBLEM FORMULATION

We assume that intensive applications have already been offloaded onto the edge server (ES), integrated with a communication base station (BS). These applications are denoted as a set of sub-tasks $\boldsymbol{X} = \{x_i\}$. The main problems and challenges can be summarized as follows:

- The tasks are placed in a wait queue to process each task separately, which leads to an increase in the service time T_e and a delay in responsiveness. Therefore, we aim to achieve $Minimize(T_e)$.
- Mobility challenges arise from how to maintain the computation services when the MD moves out of the initial ES's coverage area. Therefore, we need to measure how long the mobile device is connected to the ES and the maximum associativity ($\mathcal{A}_{E,X}$) of a task with an ES.
- When creating VMs $V = \{v_1, v_2, v_3, \ldots, v_n\}$, each sub-task is assigned to an individual VM for parallel processing, thereby reducing processing time. However, this leads to waste over the capability of the ES, along with an increase in the service cost (C_i). The tasks are stacked in the queue to wait for available VMs to receive new tasks, increasing the computation delay. Therefore, we aim to reduce the cost by the minimizing the number of VMs, namely, $Minimize(\eta_j)$.

We use process capability measures (Cp) and process performance (Pp) to monitor and verify the ES's ability to serve the offloaded task. Cp and Pp work to compare the process requirements to execute the tasks with the performance of the ES [45]. If the ES is capable of computing the offloaded task, virtual servers (i.e., VMs) are created, and the tasks are assigned to them; in turn, the value of each VM is determined (the VM's value, VMV) based on the CPU speed and size of memory allocated to it. If the virtual server (VS) is overworked, it will be incapable of serving the sub-tasks. Furthermore, the weight process capability (W-PC), which we denote by γ, is used to measure the ES's capability. Cp and Pp are observing indices for the spread of our operations compared to the specification spread. In brief, they are used to determine whether a process is capable and receive feedback. The greater the value of γ, the better; this is because γ measures the process capability to achieve defect-free work.

$$\gamma_v = \frac{1}{t_p}\omega_{vi}, \quad v_i = \{v_1, v_2, \ldots, v_n\}, \quad \forall v_i \epsilon e \tag{1}$$

$$\Upsilon_e = \sum_{i=1}^{n} \gamma_{vi}, \quad \forall e \epsilon ES \tag{2}$$

where $\boldsymbol{\omega}$ workload computation for each created VM at the edge server, and $\boldsymbol{t_p}$ is the processing time.

The process capability ratio is an ES capability measure for offloaded task computation within specified limits. The process capability analysis helps determine the ES's ability to serve the offloaded tasks within the tolerance limits.

$$\omega_{vi} = \frac{f_{vi}}{t_p} \tag{3}$$

where

$$\omega_e = \sum_{i=1}^{n} \omega_{vi}, \quad \forall v_i \epsilon e \tag{4}$$

$$When\ the\ t_p = \left(-\frac{1}{f_{vi}}\right)\mathcal{D} \tag{5}$$

where $\boldsymbol{\mathcal{D}}$ size of data, and f_{vi} a computation frequency of each VM.

Thus, the measure of the capability of edge node resources is denoted using a two-dimensional matrix, Cp, in which an element $\boldsymbol{Cp(x_i, v_i)}$ represents the computation time at the virtual server $\boldsymbol{v_i}$ for the sub-task $\boldsymbol{x_i}$. The sub-task $[x_i]$ assigned to the first v_1 And the capability of v_1 is evaluated (workload $\boldsymbol{W_v}$), is as capable of serving x_2 unless it is assigned to $\boldsymbol{v_2}$. In turn, $\boldsymbol{v_1}$ is re-evaluated for its ability to serve x_3 and if it cannot do so, $\boldsymbol{v_2}$ is evaluated; in the event that it is not able to, it is assigned to v_3. In case v_1 can serve x_3, $\boldsymbol{v_1}$ serves $\boldsymbol{x_1}$, and $\boldsymbol{x_3}$ concurrently, and so on, where ultimately $\mathbf{X_v} = \{\boldsymbol{x_{v,1}}, \boldsymbol{x_{v,2}}, \ldots, \boldsymbol{x_{v,j}}\}$.

If the measure of how much "natural variation" a procedure experiences relative to its specification limits, this enables a comparison of different processes concerning each other, after which the most effective process can be chosen. Cp and Pp are monitoring indicators for the distribution of offloaded tasks compared to current capacity and the specification of ESs.

$$Cp_e = (USL - LSL)/\Upsilon_e \tag{6}$$

$$Pp_e = (USL - LSL)/\Upsilon_e \tag{7}$$

where USL is the upper specification limit indicating the highest value that can be assigned to each VM, such as CPU and memory size, which reflects the VMV. Moreover, LSL is the lower specification limit, which indicates the lowest value that can be set for each VM. These are controlled and allocated by resource allocation management control (RAMC), responsible for assigning VM tasks in VMs. The CP for each VM is stated as follows:

$$Cp_{v_i} = (USL - LSL)/\Upsilon_v \tag{8}$$

Therefore, the scheduler must know the characteristics of each task and the task request, including variables such as data size $\mathcal{D}$, workload computation (rate of maximum processing of node) $\boldsymbol{\omega}$, required CPU frequency $\mathcal{F}$, and memory $\boldsymbol{M}$. It is possible to acquire this information through the profiling process. We assume that the profiling process retrieves information about offloaded task requirements.

Additionally, profiling is used to determine the LSL and USL of the ES, after which the LSL and USL of each created VM are calculated. If the execution requirement of the offloaded tasks falls between the limits of capabilities, then they are given priority for execution. Otherwise, they are placed in the queue, and a VM is created that conforms to their requirements, with the process carried on similarly until it terminates.

Execution. Otherwise, they are placed in the queue and a VM is created that conforms to their requirements, and so on until it is completed.

$$R_{ij} = \begin{cases} 1, & \textit{if } x_i \textit{ allocate on } V_j \\ 0, & \textit{otherwise} \end{cases} \tag{9}$$

Therefore, the service time of the sub-task x_i running on VM at the edge node V_J, denoted as $T(x_i, v_j)$, can be expressed as:

$$T_{x_i}^{V_j} = (R_{ij} * \frac{f_i}{\Upsilon_v}(x_i, v_i)) \tag{10}$$

where $\mathcal{F}_i$ is the instruction number to process for sub-task x_i. Therefore, the service time in the ES for X_i can be calculated as follows:

$$T_X^e = \sum_{i=1,j=1}^{n,m} T_{x_i}^{V_j} \tag{11}$$

In addition, to meet the computation task requirement, it is necessary to satisfy the following condition:

$$T\left(X_i, V_j\right) \le req(T_i) \tag{12}$$

The scheme of optimal resource scheduling in MEC is considered a multi-objective optimization (MOO) problem that reduces the service execution time $Minimize(T_e)$.

$$minimize \sum_{i=1}^{N} T_e, \quad e \in E \tag{13}$$

It is assumed that these tasks are related to a mobile user because the ES received the task, and the user moved out of the ES's coverage area. Furthermore, we suppose that there are no other requirements for the task; that is to say, there is no interaction between the user and the task. The network monitoring unit tracks the user, after which the nearest ES is chosen, and the output is transmitted to the user via the identified ES. The control edge cloud computing controller (CE3C) is used to find the mapping between the ESs that are closest to each other, as well as the optimal mapping of the mobile device (MD) to the ES, which is known as task associativity ($\mathcal{A}_X$). In this case, the aim is to find the greatest value of $\mathcal{A}_X$ between the ES and the task, namely, $Maximize(\mathcal{A}_X)$, which is calculated using the following equation:

$$\text{Maximize } \mathcal{A}_x = \begin{cases} 1, & \textit{when the velocity } r_i = 0 \\ \textit{otherwise}, & \dfrac{d_{(l_t - l_{t+1})R_{ij}}}{r_i} \end{cases} \tag{14}$$

where d is the distance between the location l_t and the next location l_{t+1}, and r_i is the velocity of the population i. Due to MD mobility, the location doesn't change. When $\mathcal{A}_x = 1$, this means that the velocity of particles is 0, which implies that the particles are in the same position as they were when the offloaded tasks were received. We suppose that the VM's capability is constantly checked when a new offloaded task x_i arrives at the ES, thereby ensuring the ability of the currently active VM_j to serve it before creating a new VM, which reduces the service cost. We also assume that the LSL and USL values for each VM are given. First of all, we need to calculate the capacity of the available task processing $\mu_{i,j}$ of VM_j for each corresponding task x_i [46].

$$\mu_{i,j} = p_{i,j}\lambda_i / t_{pi} \tag{15}$$

where the $LSL \le \mu_{i,j} \le USL$, $|USL, LSL| \in CP$.

Where $p_{i,j}$ is the probability VM_j in the edge, server to serve the offloaded x_i the task and λ is the task arrival rate in the edge server.

TABLE 1. The important notations used in this paper.

Notation	Definition
t_p	processing time of the task in the VM
γ_v	Weight process capability of VM
γ_e	Weight process capability of edge server (ES)
ω_{vi}	Workload computation of VM
ω_e	Workload computation of ES
DPT	Dynamic programming technique
$r_i(t)$	Current velocity of the particle
$L_i(t)$	Current position of the particle
VMv	VM value
VMc	Cost of each VM
ρ_t^*	Price of each VM per time
η_j	Number of VMS
TPT	Task processing time
$T_{e,X}$	Time of the processing task at the edge server
$\mathcal{A}_{E,X}$	Task associativity
$e_{i_{cost}}$	Cost of the edge server
α	An acceleration coefficient
f_{vi}	Computation frequency of VM
Cp_e , Cp_v	Process capability of ES, and VM respectively
Pp_e,Pp_v	Process performance of ES, and VM respectively
R_{ij}	Resource allocation of the task x_i In the VM v_j
$\mathcal{F}_i$	Instructions number to the process of task X
T_X^e, T_X	Services time of the offloaded task X_i On ES
T_x^v	Services time of the sub-task x_i On VM
$\mathcal{D}$	Uploaded massive real-time tasks
$\mu_{i,j}$	VM capacity of available task processing
USL & LSL	Upper specification limit, and
$p_{i,j}$	Probability VM_j In the edge server to serve the offloaded x_i , lowest value that can be set for each VM
L^*	Best position of the particle

We assume that the λ is fixed. In practically, the available task processing capacity of the VM_j is greater than task request rate $\theta_{i,j}$ to satisfy task QoS requirements.

$$\theta_{i,j} = p_{i,j}\lambda_i,\ \text{where } \mu_{i,j} \geq \theta_{i,j} \tag{16}$$

Therefore, the sum of the processing capacities of all VMs in the ES is the available task processing capacity of the ES.

$$\mu_j = \sum\nolimits_{j=1}^{n} \mu_{i,j} \tag{17}$$

We aim to reduce the process of creating VMs, as well as the number of VMs used to serve an offloaded task, to facilitate a reduction of service cost by maintaining resources, namely, *Minimize*(η_j):

$$\eta_j = \begin{cases} 1, & \dfrac{\theta_{i,j}}{\mu_j} \leq 1 \\ round\left(\dfrac{\theta_{i,j}}{\mu_j}\right), & \dfrac{\theta_{i,j}}{\mu_j} \geq 1 \end{cases} \tag{18}$$

Unlike single-objective optimization problems, whose aim is only to achieve the single most efficient solution, MOO problems allow a set of factors to be regarded as a set of the Pareto-optimal solutions, which represents the options between these objectives. Table 1 provides an overview of the important notations used in this research.

VI. MULTI OBJECTIVE HYBRID ACCELERATED PARTICLE SWARM OPTIMIZATION AND DYNAMIC PROGRAM (MOAPSO-DP)

The proposed model uses three algorithms: first, the non-preemptive priority algorithm; second, accelerated particle swarm optimization (APSO); and third, dynamic programming (DP) for the knapsack problem.

The first algorithm used in the proposed model is the non-preemptive priority algorithm. This is used to sort the offloaded tasks and process them according to the priority assigned to each task. In this way, any task that has started processing is prevented from being transferred to another resource, thereby increasing the task's associativity with the edge server (ES) to which it has been offloaded (regardless of whether the mobile device (MD) moves to another ES coverage area). This means that task execution is not interrupted until the completion of processing. Following this, the second algorithm – the APSO algorithm –assigns these tasks to the VMs according to priority to start processing while minimizing processing time. In turn, DP is used to perform resource balancing across the VMs to reduce the service cost.

The assumption is made that tasks are offloaded to the ES. The density of a task is partitioned into sub-tasks x_i, where $x_i \in \boldsymbol{X}, \mathrm{X}_\upsilon = \left\{x_{\upsilon,1}, x_{\upsilon,2}, \ldots, x_{v,j}\right\}$. Therefore, it is necessary to make a tradeoff between makespan and optimal resource utilization, which will reduce cost. The primary process involved in the model is divided into three steps: task partitioning, scheduling optimization, and migration only result from the ES to another, as shown in Figure 2. The scheduling of the task in relation to the available edge computing resources (ECR) can be modeled as follows:

When tasks are offloaded to the ES, the solver in the ES partitions each task into sub-tasks $\boldsymbol{X} = \{x_\mathrm{i}\}$. This enables concurrent processing for the VMs to accommodate the task without waste over the capability. Accordingly, the VMs are filled as stated before: to process the sub-tasks on the VMs one by one, thereby reducing service cost and service time due to concurrent processing. This yields $\mathrm{X}_\upsilon = \left\{x_{\upsilon,1}, x_{\upsilon,2}, \ldots, x_{v,i}\right\}$, where x_υ is the task indicator vector of the υ^{th} VM, and $x_{v,i}$ is the indicator of the i^{th} sub-task. If the VM υ is capable of serving the sub-tasks i, then $x_{v,i} = 1$; otherwise, $x_{v,i} = 0$. The task details for all VMs can be expressed as a task indicator matrix $X(i,j) = \left\{X_{1,\mathrm{v}_j}, X_{2,\mathrm{v}_j}, \ldots, X_{\mathrm{n},\mathrm{v}_j}\right\}$, where the υ^{th} a row is the task indicator vector of the ES υ.

A. ACCELERATED PARTICLE SWARM OPTIMIZATION (APSO)

In the proposed model, we use the simplified model of the PSO algorithm, which is known as accelerated PSO (APSO) [8]. Both the global best g and the individual best x are used in standard particle swarm optimization. The individual best is primarily used to increase the diversity of the quality solutions; however, this diversity can be simulated with randomness. The use of individual best solutions is not essential unless the optimization problem in question is highly nonlinear and multimodal.

Using APSO was to facilitate efficient resource utilization and resource management in the ES as the primary objective, reduce service cost to the customer as a secondary objective, and increase the task's associativity to the edge server as an auxiliary objective. Therefore, the use of APSO was expected to yield promising results, as indicated by a comparison of the simulation results to other algorithms.

For our model, a key objective of resource scheduling was to minimize computation latency for tasks offloaded to the ES and maintain resources to serve the largest number of tasks at the lowest cost. We used MOO algorithms based on APSO to pursue these objectives, which were represented in the minimization of the response time of the task T_i and the conservation of resources V_i. This can be expressed using the matrix $t_i(X_i, V_j)$. In this study, we used the value of the fitness function for all objectives instead of using the concept of the Pareto set[47]. This is because the Pareto approach uses multiple comparisons to find dominant solutions, which takes longer. The weight or the value for each objective's method provided better solutions by relying on only weight lists. In this method, we establish the weight for each solution based on the values of the objectives in the MOAPSO algorithm to select the appropriate VM (V_j) for each task (x_i), as shown in Algorithm 1.

Our objectives are to reduce the task processing time (TPT) at the ES, $T_{e,X}$ (minimize TPT), increase the task associativity $(\mathcal{A}_{E,X})$ of a task with an ES (highest AT), and minimize the VM (V_i, x) cost, which is reflected in the overall cost of the ES (minimize $\boldsymbol{e}_{i_{cost}}$). The particles in this work are represented as VMs, while the values of best fitness are calculated according to the three factors: minimize the task processing time (TPT), maximize task associativity (TA), and minimize $\boldsymbol{e}_{i_{cost}}$.

The TPT is calculated in Equation 9. The cost of each ES depends on the number of VMs, as calculated in Equation 12, while the cost of each VM (VMc) depends on the VM's value (VMv) value. The value of the VM is based on the VM's CPU and memory, which is retrieved automatically from the VM profile; it is defined as a weight w. Therefore, the edge cost $\boldsymbol{e_{cost}}$ is calculated as follows:

$$VMc_{j,x_i} = T_j * \text{VMv}_j \tag{19}$$

$$\text{Minimize } \boldsymbol{e_{i_{cost}}} = \eta(VMc_{j,x_i})\rho_t^*, \quad \forall v_j \epsilon e_i \tag{20}$$

where η represents the number of VMs that were created in each ES, which was also calculated in function (12), Additionally, ρ_t^* represents the price of each VM per time.

Algorithm 1 MOAPSO Task Scheduling

```
1: input: velocity values
2: Output: Update velocity values, and updated particles position
3: Set $\delta_1, \delta_2 = rand[0, 1]$, $\alpha_1, \alpha_2 = 2.05$
4: Procedure MOPSO (X, VMS)
5:    for $x \in X$ do
6:       for $v \in vm$ do
7:          TPT $(x, v)$ =COMPUTE_TPT $(x, v)$ //Eq#13
8:          TA $(x, v)$= COMPUTE_TA $(x, v)$ //Eq#14
9:          VMC $(x)$ = COMPUTE_VMC $(x)$ ) //Eq#19
10:      end for
11:   end for
12: INITIALIZE (VMs, Velocity, Position, vbest,)
13: for $x_i \in X$ do
14:  for $v_j \in VM$ do
15:   f=FITENSS (TPT, TA, VMC)
16:   If $v_j = v$best //Update velocity
17:      $v$best =EVALUATE(f)
18:      Velocity $(v_j) -= \delta * \alpha$; $//\delta randomnumber$
19:   Else
20:      Velocity $(v_j) += \delta * \alpha$; $//\alpha accelerationcoefficients$
21:   End If
22   Maxvelocity1=get _max1($v_j$, Velocity values) //Update position
23:   Maxvelocity2=get _max1($v_j$, Velocity values)
24:   Swap($v_j$[Maxvelocity1], $v_j$[Maxvelocity2])
25:      end for
26:   end for
27: return $v$best
28: end Procedure
```

Regarding the issue of calculating the associativity AT ($\mathcal{A}_{E,X}$), we assume that each particle $\boldsymbol{VM}$ in the swarm behavior has two main attributes: a position (L), which specifies the suggested location of the ES related to the tasks, and a velocity (r). The APSO algorithm uses a movement that changes a particle's position in each iteration. The APSO updates a particle $\boldsymbol{L_i}$ at time t is shown in the following equation [48]:

$$L_i\,(t) = L_i\,(t-1) + r_i(t) \tag{21}$$

where $\boldsymbol{L_i}\,(\boldsymbol{t})$ is the current position and $\boldsymbol{L_i}\,(\boldsymbol{t}-\boldsymbol{1})$ is the next position. In addition, the velocity of particle i at time t is given in the following equation [48]:

$$r_i(t) = \theta r_i(t-1) + \delta\alpha(vbest_j - l_i(t)) \tag{22}$$

where the $\boldsymbol{\theta}$ indicates the direction, δ is the random number between [0,1], $\boldsymbol{\alpha}$ is an acceleration coefficient. The $vbest$ is best position of a particle, which is a VM.

We assume that these tasks are related to an MD user as the ES receives the task and changes their location due to mobility. Additionally, considering that there are no other requirements for the tasks, there is no interaction between the user and the offloaded tasks. Further, we assume that the processing occurs on the ES that receives the offloaded task; the first time, the task is associated with a VM, and immediately, scheduling and processing procedures are initiated. This operation leads to an increase in the value of $\mathcal{A}_x$. Otherwise, the recent ES dispatches the offloaded task to the nearest ES. The user is tracked by the ES and the network monitoring unit; the results are forwarded through the adjacent ES. If the user is outside the range of the recent ES and forwards the offloaded task to another ES to be processed, this increases the transfer task cost and processing time. Therefore, we aim to increase the associativity $\mathcal{A}_x$ this ensures that the task is not transferred to the nearest user server except for when the recent ES cannot undertake task processing. This is considered a new contribution of the present study to the literature.

Dynamic programming (DP) was applied to minimize the cost of resource utilization and minimize the problem of wasting ES capabilities, dynamic programming (DP) was applied. Specifically, the DP technique was used to solve the knapsack problem. DP is useful for solving overlapping subproblems, which means that it must rely on a precedent value to search for a better solution. However, the DP technique depends on the situation of the current particle and the VM's value to find a better solution.

Algorithm 1 shows this study's novel scheduling algorithm for handling offloaded tasks inside the ES based on the MOAPSO-DP algorithm. The primary process in this algorithm is dedicated to the calculation of multi-objectives. The MOAPSO-DP algorithm is applied to optimize task scheduling and evaluate the three objectives, namely, to calculate the task processing time (TPT), task associativity (TA), and cost of each VM (VMC). This will enable selecting the optimal solution for the offloaded tasks in each VM using the COMPUTETPT, COMPUTETA, and COMPUTEVMC functions. These three functions apply Equations 11, 14, and 19, respectively. Additionally, the MOAPSO-DP algorithm contains the main procedures of APSO, but several objectives are used instead of one goal.

The primary process in Algorithm 1 is concerned with calculating multi-objectives. The MOAPSO-DP Algorithm 3 is applied to optimize task scheduling and evaluate the three objectives: to calculate the task processing time (TPT), task associativity (TA), and cost of each VM to select the best solution. In dealing with a MOO problem, we can either integrate every objective into a single objective or consider each function separately. Therefore, in this research, DP was used to solve the knapsack problem, thereby balancing resource allocation, improving resource utilization, and solving the waste over the capability of each VM in the ES, as shown in Algorithm 5. solving the waste over the capability of

Algorithm 2 FITNESS Algorithm

```
1: Procedure FITNESS (TPT, VMC, TA)
2:     INITIALIZE (RP) //Repetition & Priority
3:     $v_i \leftarrow$ Value1=Min (TPT)
4:     $v_i \leftarrow$ Value2= Max (TA)
5:     $v_i \leftarrow$ Value3= Min (VMC)
6:     for $v_i \in VMs$ do
7:     DP= DP ($v_i$)
8:     end for
9:   return DP ($v$)
10: end Procedure
```

In the MOAPSO algorithm, the fitness function is invoked to evaluate the objectives, as shown in Algorithm 2. The objectives are represented by a two-dimensional matrix ($\boldsymbol{x}$, $\boldsymbol{v}$). Algorithm 2 also shows the process used to compute the value of the fitness function for the objectives. The value of each particle (i.e., VM) is computed for each objective, and the best value is selected for each particle. Consequently, the particle with the greatest weight value from the corresponding three objectives is chosen as the best solution.

Algorithm 3 MOAPSO-DP (X, VMs)

```
1: input: Set of Tasks $\{X_1, X_2, \ldots., X_n\}$, set of resources $\{VM_1, VM_2, \ldots., VM_n\} \in ES$
2:         Set of subtasks $x_i \in X_i$, and set $v_j \in VM_j$
3: Output: $maping(x_i, v_j)$ // the best solution to allocate $X_i$ over VM$_j$
4: For $i = 0$ to $N^X$ // number of tasks
5:         $n^{VM} = \frac{\mu_k^{ES}}{X_i^n \mu_i^X}$ // create VM per task on demand
6:  While not Reach $n$ do // number of iterations
7:  MOAPSO ($X_i$, $V_j$)
8:    For J=0 to n // number of VM
9:     DP ($x_i$, $v_j$)
10   End For
11: Repeat
```

In more detail, for TPT, TA, and VMC, the task selects an appropriate VM with the greatest weight value depending on the balancing method. Weights are assigned because the objectives are different: the first objective is TPT and the second is VMC, the best solution is Min, and the third objective is TA is Max. This is an instance of the bounded knapsack problem (BKP), classified as an NP-hard combinatorial optimization problem. The BKP can be stated as follows:

$$minimize \sum\nolimits_{i=1}^{n} \upsilon_i X_i \tag{23}$$

$$subject\ to \mu^e = \sum\nolimits_{i=1}^{n} \mu_i^v, v_i \in \{0, 1, 2, \ldots., n\} \tag{24}$$

$$\text{Constraints: } 0 \leq \upsilon_i X_i \leq \mu_i^v$$

$$L^* = max(\mu_i^v) \tag{25}$$

where L^* represents the optimal position. In this study, the DP technique was used to solve the BKP. In particular, to identify the available VMs optimally, we used the task allocation process shown in Algorithm 4:

B. DYNAMIC PROGRAMMING

Dynamic programming (DP) [49] is a technique that is used to solve overlapping sub-problems. To design a DP algorithm for a knapsack problem, it is first necessary to derive a recurrence relation to express a solution as an instance of the knapsack problem. In DP, each sub-problem is solved separately, the outputs are stored in a table, and the table is subsequently used to solve the original problem. The instance is regarded as a problem defined by the first x_i tasks, where $1 \leq i \leq N$, within the available task processing capacity μ_j of the VM$_j$ for each particle $v_1 \ldots., v_i$ and VM capacity v_j, where $1 \leq j \leq \mu_j$; and to determine the resource requirements of $VM_j(t)$, which are represented as time requirement, type of resources, and amount of resources. The amount of resources can be calculated based on the start time of the request $\boldsymbol{time_s}$, duration time $\boldsymbol{time_d}$, and resource type $\boldsymbol{res_n}$.

We calculated the utilization of the resource $\boldsymbol{R_{ij}}$ of each node VM_j(t), by:

$$R_u(t) = \frac{1}{\mu_j} \sum\nolimits_{n=1}^{n} R_{ij}(t) .r_n \tag{26}$$

where R_{ij} is a binary variable to determine whether v_i is allocated by the task at the time instant t or not. In addition, r_n resource amount $r_n = \{time_s, time_d, res_n\}$. According to the computation offloading, the number of tasks offloaded on $v_i(t)$ at time instant t is calculated by:

$$VM(t) = \sum\nolimits_{j=1}^{n} R_j(t) \tag{27}$$

In this study's DP approach to the problem, the task $\boldsymbol{X}$ was split into sub-tasks $\boldsymbol{x_i}$ to fit each VM under the following constraint:

$$0 \leq \sum \mu_i^x \leq \mu_j^v \tag{28}$$

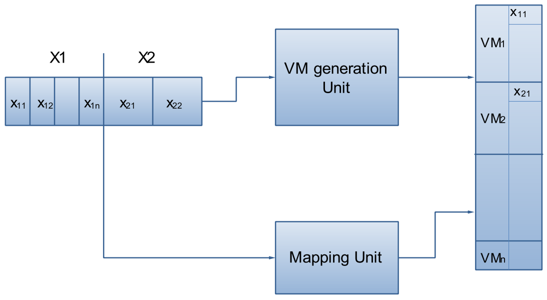


FIGURE 3. Partition of the tasks and accommodates their by VMs.

Let $\boldsymbol{table}[\boldsymbol{i}, \boldsymbol{j}]$ be the optimal solution of this instance (i.e., the weight (capacity)) of the most valuable of the first sub-tasks $\boldsymbol{x_i}$ that fit into $\boldsymbol{VM_j}$. It is possible to divide all the sub-tasks $\boldsymbol{x_i}$ that fit into $\boldsymbol{VM_j}$ into two categories: first, sub-tasks that do not include the $\boldsymbol{x_i^{th}}$; and sub-tasks that include the $\boldsymbol{x_i^{th}}$. This leads to the iteration shown in Algorithm 5.

As shown in Algorithm 5, to find Table $[N, Weight]$, the maximal value of a sub-task of VM under the conditions of the two boundaries for the VM are:

- The VM equals zero when there is no task allocated in it (i.e. x_i= 0).

$$Table\ [0,j] = 0\ for\ v_j \geq 0$$

Algorithm 4 Task Allocation (X, VMs)

```
1: Initialize: $x_n \leftarrow X_n$, $\mu^v \leftarrow \mu^V$ [$Capacity$]
2: Start at location $Table[x, v]$
3: While the capacity is greater than 0, do 4:     If $Table[x_n, v] = Table[x_{n-1}, v]$ then
5:     Subtask $x_n$ has not been available in the optimal solution
6:     Else
7:     Subtask $x_n$ has been available in the optimal solution
8:     V==MOAPSO-DP $(x_n, v)$
8:     mapping $(x_n, v)$ and start process subtask $x_n$
9:     Move one row up to $x_{n-1}$
```

- The VM equals zero when its weight has no value (i.e. $v_j = 0$) because no task can be allocated in it.

$$Table\ [i, 0] = 0\ for\ x_i \geq 0$$

Algorithm 5 Dynamic Programming DP

```
1: Initialize:
($Weights[1 \ldots N]$, $Values[1 \ldots N]$, $Table[0 \ldots N, 0 \ldots Capacity]$)
2: Input: Weight array include the weights(capacity) of all task
3:        Values array include the priority values of all task
4:        Table Array used to save the results
5: Output: The final value of Table array ($Table[N, Capacity]$) contains the solution for the given weight
6: $for\ i = 0\ to\ N\ do$
7: $for\ j = 0 to\ Capacity$
8:  If $\mu_i^v < \mu_i^x$ then
9:    $Table[i, j] \longleftarrow Table[i-1, j]$          Cannot fit the $x_i^{th}$ subtask
10:   Else
11:   $Table\ [i, j] \longleftarrow maximum\ Table[i-1, j]$   Do not use the $x_i^{th}$ subtask
12:        $And$
13:   $p_i + Table\left[i-1, j - weight_i\right]$       Use the $x_i^{th}$ item
14:  Return $Table[N, Capacity]$
```

C. NON-PREEMPTIVE PRIORITY BASED SCHEDULING (NPP)

We used a non-pre-emptive priority-scheduling algorithm to sort the offloaded tasks according to the priority assigned to each task. Thus, this prevented any task that had started its processing from being transferred to another resource, thereby increasing the task's associativity with the ES, which has been offloaded regardless of whether the MD moved to another ES coverage area, which means not interrupting the execution of the task until the completion of its processing. In Algorithm 6, we sorted the processing of offloaded tasks based on the given priority to each task and the arrival time of each task to the edge server. Additionally, we calculated the processing time and waiting time for each task.

VII. SIMULATION AND RESULTS ANALYSIS

CloudSim [50] was used to evaluate the proposed method, MOAPSO-DP. CloudSim is an open-source package offering modules to emulate cloud environments with flexibility to modify the simulation by adding modules depending on the desired design. Specifically, we applied our model using CloudSim 3.0.3. CloudSim is valuable as a universal tool and simulation framework that permits the modeling and simulation of cloud computing infrastructures and services [50]. It includes several libraries and supports different functionalities, including events processing and queuing, creating cloud system objects (e.g., data centers, services, virtual machines, hosts, and brokers), and communication between modules.

TABLE 2. Description of edge server.

Parameter	Value
No. VMs	Depending on the no of tasks and task capacity Eq.18
No. tasks	1 - 1000
CPU MIPS	(1,500 – 3,000) MIPS
Storage	2 TB
RAM size	(32,000-64,000) MB
Bandwidth	(500-2024) MB

Our proposed MOAPSO-DP integrated three algorithms (non-preemptive priority algorithm, APSO, and programming (DP) for the knapsack problem). The APSO algorithm is considered one of the optimal algorithms for managing resource allocation, which evolved from the PSO; its pros are its quality and effectiveness, efficient, high-quality solution, and local exploitation capability. In the APSO algorithm is not essential to use individual best solutions unless the optimization problem in question is highly nonlinear and multimodal. Therefore, DP reduces the service costs by balancing resources across VMs.

Many studies have been undertaken to develop scheduling algorithms, including studies using genetic algorithms (GA) [51], PSO algorithms [52], APSO algorithms [8], and hybrid PSO-GA algorithms [53]. In the simulation for this research, we compared the proposed MOAPSO-DP algorithm to each of these state-of-the-art alternatives.

A. SIMULATION SETTING

To evaluate the impact of the proposed method, MOAPSO-DP, on the resource allocation problem in the mobile edge computing (MEC) context, we performed a simulation study with the parameters listed in the following tables: A description of the edge server (ES) is given in Table 2, a description of the virtual machines (VM) in Table 3, and a description of the tasks in Table 4. The number of VMs depends on the number and capacity of tasks arriving at the ES. In addition, VMs are created based on the number of incoming tasks, their characteristics, and whether the tasks belong to one or many users. These factors are reflected in the following equation:

$$0 < \eta_{VM_j} \leq \frac{\mu_k^{ES}}{X_i^n \mu_i^X}, \quad \exists VM_j \in ES_k \tag{29}$$

$$0 < \eta_{v_j} \leq \frac{\mu_j^{vm}}{x_i^n \mu_i^x}, \quad \exists x_i \in X_i\ and\ v_j \in VM_j \tag{30}$$

More specifically, the experimental simulation model included three edge nodes and multiple mobile devices (MD).

Algorithm 6 Non-Preemptive Priority Based Scheduling

```
1: Input: number of tasks, duration time $time_d$,arrival $time_a$, a priority of each task
2: Output: scheduling tasks depend on the priority
3: Initialize waiting $time_r$, completion time $time_c$
4: Sort the Tasks according to the priorit0079
5: For i=0 to number of tasks
6:     $Tp\,(x_i) \leftarrow time_a(x_i) + time_d\,(x_i)$   // processing time of $x_0$
7:     $Tp\,(x_{i+1}) \leftarrow time_a(x_{i+1}) + Tp\,(x_i)$   // processing time of $x_{i+1}$
8:     $time_c\,(x_i) \leftarrow Tp\,(x_i) - time_a(x_i)$   // completion time
9:     $time_w(x_i) = time_c(x_i) - time_d\,(x_i)$ // waiting time
10:      End For
11: $\text{Highest}_{\text{Priority}(x_i)} \leftarrow time_a\,(x_i)$
12: Avg $(time_w(x_i))$
13: Avg $(Tp\,(x_i))$
```

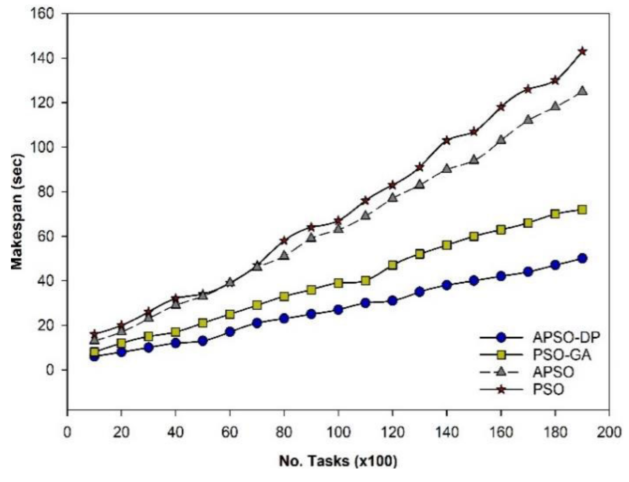


FIGURE 4. Execution time per task.

Assuming the VMs in each ES provided edge cloud services, we simulated many offloaded tasks using data from the LCG dataset [54]. This dataset consists of approximately 200,000 tasks, offering a complete description of the tasks served, such as offloaded time, MD ID, compute task name, and task runtime. In this study's simulation, the LCG dataset was used because it includes all the types of tasks relevant to what is required in the proposed model simulation.

The simulation system contained three ES nodes with predefined specifications. The number of VMs was created depending on the offloaded task requirements. The data size of each task ranged from 2MB to 4MB. The performance analyses were conducted over computation offloading activities with different tasks (from 20 to 210) and specific parameters, as shown in Table 4.

TABLE 3. Description of VMs.

Parameter	Value
CPU MIPS	2400
Storage size	40,000MB
RAM size	1024
No. VMs	42
VM Price	20 cents

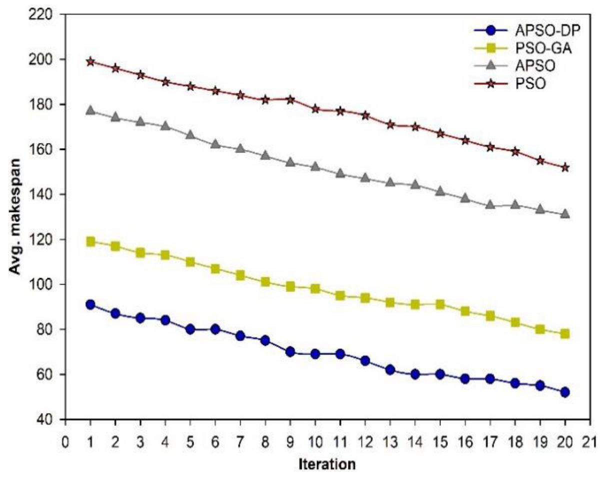


FIGURE 5. Execution time per iteration.

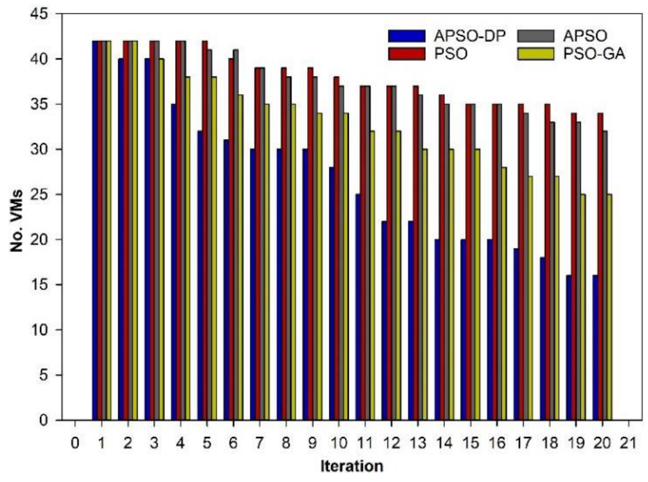


FIGURE 6. Number of VMs per iteration.

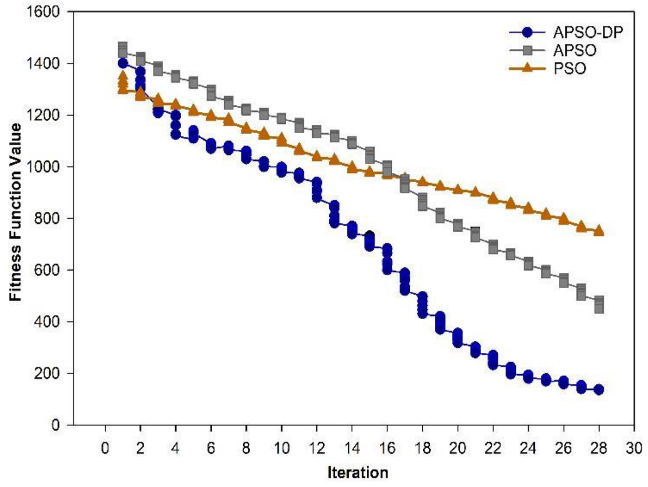
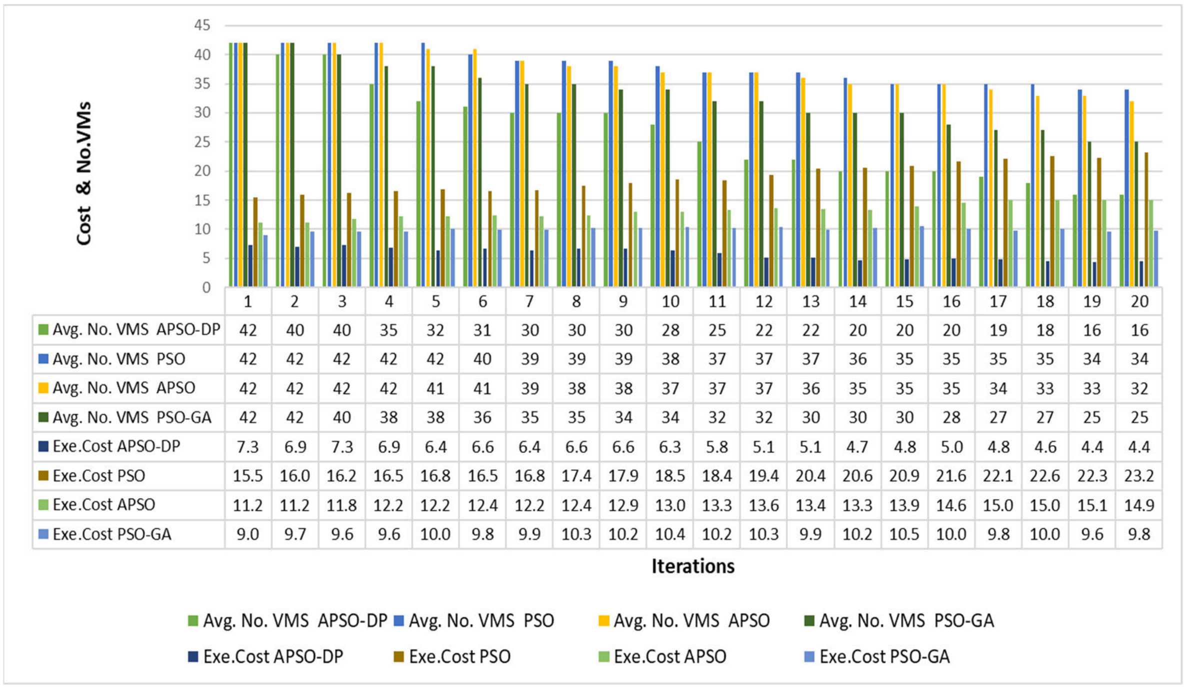


FIGURE 7. Fitness function.

Each experiment was run ten times, and the start submitted 100 tasks. The acceleration coefficient (α) was set



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Avg. No. VMS APSO-DP	42	40	40	35	32	31	30	30	30	28	25	22	22	20	20	20	19	18	16	16
Avg. No. VMS PSO	42	42	42	42	42	40	39	39	39	38	37	37	37	36	35	35	35	35	34	34
Avg. No. VMS APSO	42	42	42	42	41	41	39	38	38	37	37	37	36	35	35	35	34	33	33	32
Avg. No. VMS PSO-GA	42	42	40	38	38	36	35	35	34	34	32	32	30	30	30	28	27	27	25	25
Exe.Cost APSO-DP	7.3	6.9	7.3	6.9	6.4	6.6	6.4	6.6	6.6	6.3	5.8	5.1	5.1	4.7	4.8	5.0	4.8	4.6	4.4	4.4
Exe.Cost PSO	15.5	16.0	16.2	16.5	16.8	16.5	16.8	17.4	17.9	18.5	18.4	19.4	20.4	20.6	20.9	21.6	22.1	22.6	22.3	23.2
Exe.Cost APSO	11.2	11.2	11.8	12.2	12.2	12.4	12.2	12.4	12.9	13.0	13.3	13.6	13.4	13.3	13.9	14.6	15.0	15.0	15.1	14.9
Exe.Cost PSO-GA	9.0	9.7	9.6	9.6	10.0	9.8	9.9	10.3	10.2	10.4	10.2	10.3	9.9	10.2	10.5	10.0	9.8	10.0	9.6	9.8

FIGURE 8. Cost per the number of VMs.

at 2.05 based on the research undertaken by [47], which is because it is associated with promising results. In this simulation, we measured the following parameters:

B. REDUCING THE COMPUTATION TIME OF THE TASKS (PROCESSING TIME)

In the simulation, we measured the makespan in seconds. Figure 4 shows the differential of the time of the execution tasks based on the number of tasks.

As all the tasks in the data set numbering 200 thousand are offloaded in batches, we fixed the number of VMs at 40, with 20 epochs. We noticed that when the number of tasks is few, the difference in execution time between the proposed model and other models is close. With the increase of the offloaded tasks, the difference in execution time gradually increases according to the number of tasks. Nevertheless, the increase in the proposed method is slight, yet the improvement rate of the APSO-GA algorithm is approximately 28%. Furthermore, when offloading tasks as one batch and the VMs are created dynamically based on the absorption of offloaded tasks, with 20 epochs.

TABLE 4. Description of tasks.

Parameter	Value
Task number	1000-200,000
Length (MB)	100-3000
MIPS	500-5000

The results are shown in Figure 4. Also, we compared the performance of the proposed algorithm to the standard PSO, APSO, and PSO-GA algorithms. Figure 5 shows the execution times of each algorithm based on the iterations. Significantly, the proposed algorithm achieved the lowest execution time.

Figure 6 shows the number of VMs for each iteration. For the first iteration, the number of VMs was 42. The proposed algorithm's performance indicates that for the last iteration, it used a smaller number of VMs and still maintained the lowest execution time compared to the other three algorithms (see Figure 5). As shown in Figure 7, the fitness function values demonstrate that the proposed algorithm outperformed two other APSO and PSO algorithms with a minimum value in the last iteration and an improvement of 30%.

C. OPTIMIZING THE PROCESSING COST (COST OF SERVICES)

The cost of service comparison between MOAPSO-DP and the standard PSO, APSO, and PSO-GA algorithms is shown in Figure 8. Notably, MOAPSO-DP-based task scheduling achieved lower costs compared to the other algorithms. This can be attributed to the algorithm's ability to balance the load, which reduced the number of VMs used. This directly reduces the costs, as shown in the table underneath Figure 8 (the fee for each VM is estimated at approximately 0.2 cents per 60 seconds). Figure 8 shows the service cost for each algorithm with a different number of VMs, where the number of tasks was fixed at 50. As shown in Figure 9, when the number of tasks was changed, the cost increased gradually with the number of tasks; notably, the tasks were distributed to a sufficient number of VMs to minimize processing time.

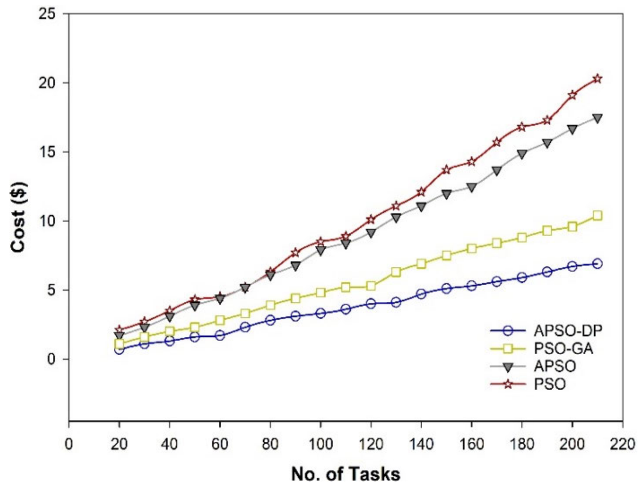


FIGURE 9. Cost per the number of tasks.

The reduction of VM numbers has effects on the cost of service. Figure 10 showed the proposed algorithm's performance per iteration when the number of tasks was fixed at 50. Figure 10 also shows the proposed algorithm at a lower cost based on Knapsack's algorithm, which uses the principle of efficient resource utilization and reduces VMs capability wastage. The cost decreased gradually, and the proposed algorithm achieved a lower cost compared to the other algorithms.

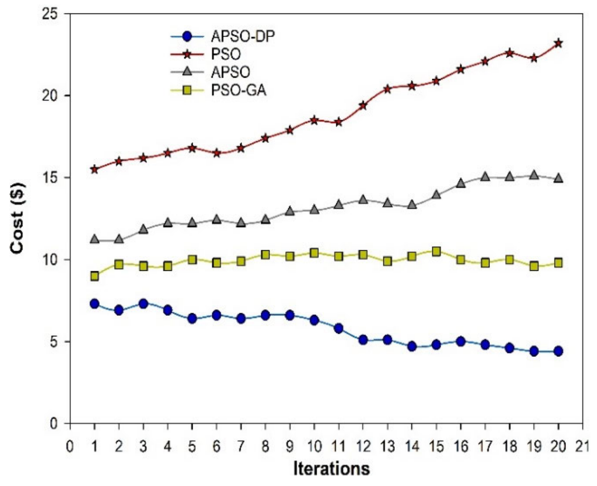


FIGURE 10. Cost per iterations.

D. RESOURCE UTILIZATION

Figure 11 showed the level of resource utilization when the VMs were created to handle offloaded tasks; the number of tasks ranged from 20 to 210. As the figure indicates, the proposed algorithm outperformed the other algorithms in resource utilization and achieved remarkable efficiency. The simulation results demonstrated that VMs were reduced when the tasks were sent to a smaller number of VMs. The number of VMs started decreasing from 42 VMs until

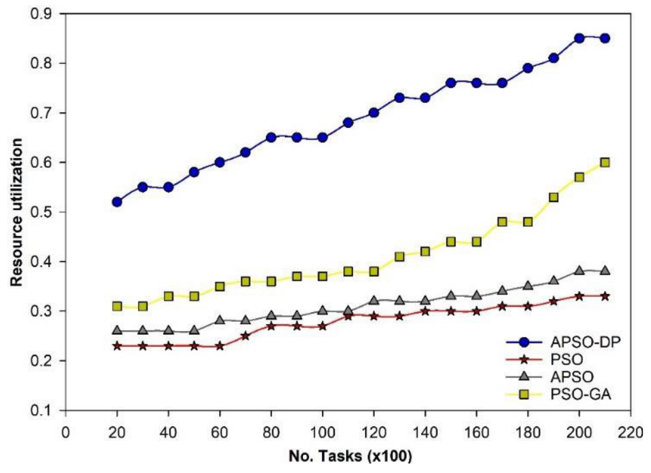


FIGURE 11. Resource utilization.

reaching the last iteration of 16 VMs, directly reflected in the optimization of the resource utilization rate, which reached 85%. We noticed that resource utilization was inversely proportional to the cost of service; in particular, the greater the resource utilization, the lower the cost of service.

E. AVERAGE WAITING TIME

Average waiting time was measured by computing the difference between the time a task was offloaded to the ES and the starting time of task execution. Figure 12 shows that the waiting time of the proposed algorithm was lower than the other algorithms. The number of tasks was fixed at 50, but the number of VMs changed in each iteration based on the resource balancing method, thereby eliminating the wastage of resource capabilities. Figure 13 shows the average waiting time based on the number of tasks when VMs were fixed at 42 VMs. The waiting time was affected by the number of tasks offloaded at the ES, as the waiting time gradually increased based on the number of tasks.

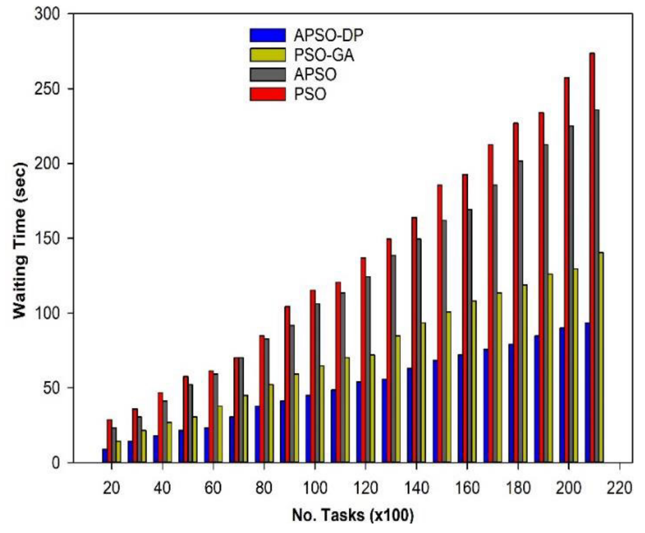


FIGURE 12. Waiting time average per tasks number.

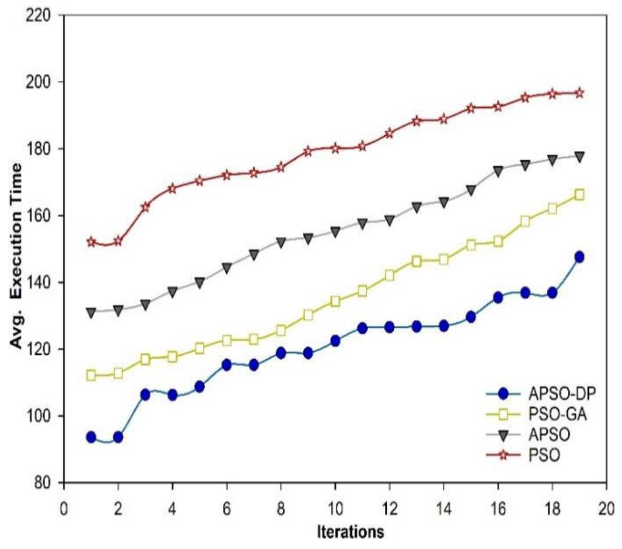


FIGURE 13. **Waiting time average per iteration when tasks number is fixed at 50 tasks.**

VIII. CONCLUSION

Mobile edge computing (MEC) systems provide integrated services close to end-users to achieve QoS and quality of QoE. This study examined how to jointly optimize resource allocation when offloading tasks from mobile devices (MD) to edge servers (ES) in MEC systems, thereby minimizing computing time and cost of service. In this case, the tasks can be processed locally and offloaded to the MEC server. The main insight from this study is that the offloaded tasks are scheduled to the VMs in the ES to minimize computing time, service cost, the waste over the capability of the ***ES***, and the maximum associativity ($\mathcal{AE}$) of a task with an ES. This paper introduced MOAPSO-DP as a novel method based on resource allocation management and offloaded task scheduling to achieve a state-of-the-art outcome in MEC. Three algorithms were used in the proposed model: non-pre-emptive priority algorithm, APSO, and dynamic programming (DP) for the knapsack problem (KP). The experimental comparison of the proposed model to PSO-GA demonstrated that the proposed model outperformed it by reducing task makespan by 30% and increasing resource utilization by 29%. Additionally, the proposed method lowered service cost, waiting time, and improvements in fitness function value compared to the PSO, APSO, and PSO-GA.

REFERENCES

[1] M. J. A. S. M. C. Satyanarayanan and C. Review, "Mobile computing: The next decade," vol. 15, no. 2, pp. 2–10, 2011.

[2] G. H. Forman and J. Zahorjan, "The challenges of mobile computing," *Commun. ACM*, vol. 36, no. 7, pp. 75–84, 1993.

[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st, Ed., MCC workshop Mobile Cloud Comput.*, 2012, pp. 13–16.

[4] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—A key technology towards 5G," *ETSI White Paper*, vol. 11, pp. 1–16, Sep. 2015.

[5] T. Alfakih, M. M. Hassan, A. Gumaei, C. Savaglio, and G. Fortino, "Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on SARSA," *IEEE Access*, vol. 8, pp. 54074–54084, 2020.

[6] J. Dolezal, Z. Becvar, and T. Zeman, "Performance evaluation of computation offloading from mobile device to the edge of mobile network," in *Proc. Standards Commun. Netw. (CSCN)*, Oct. 2016, pp. 1–7.

[7] C. Luo, S. Salinas, M. Li, and P. Li, "Energy-efficient autonomic offloading in mobile edge computing," in *Proc. IEEE 15th Int. Conf. Dependable, Autonomic Secure Comput., 15th Int. Conf. Pervasive Intell. Comput., 3rd Int. Conf. Big Data Intell. Comput. Cyber Sci. Technol. Congress(DASC/PiCom/DataCom/CyberSciTech)*, Nov. 2017, pp. 581–588.

[8] X.-S. Yang, S. Deb, and S. Fong, "Accelerated particle swarm optimization and support vector machine for business optimization and applications," in *Proc. Int. Conf. Netw. Digit. Technol.* Berlin, Germany: Springer, 2011, pp. 53–66.

[9] T. Zhao, S. Zhou, X. Guo, Y. Zhao, and Z. Niu, "A cooperative scheduling scheme of local cloud and internet cloud for delay-aware mobile cloud computing," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Dec. 2015, pp. 1–6.

[10] X. Guo, R. Singh, T. Zhao, and Z. Niu, "An index based task assignment policy for achieving optimal power-delay tradeoff in edge cloud systems," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–7.

[11] V. Di Valerio and F. Lo Presti, "Optimal virtual machines allocation in mobile femto-cloud computing: An MDP approach," in *Proc. IEEE Wireless Commun. Netw. Conf. Workshops (WCNCW)*, Apr. 2014, pp. 7–11.

[12] S. M. S. Tanzil, O. N. Gharehshiran, and V. Krishnamurthy, "Femto-cloud formation: A coalitional game-theoretic approach," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2015, pp. 1–6.

[13] J. Oueis, E. Calvanese-Strinati, A. De Domenico, and S. Barbarossa, "On the impact of backhaul network on distributed cloud computing," in *Proc. IEEE Wireless Commun. Netw. Conf. Workshops (WCNCW)*, Apr. 2014, pp. 12–17.

[14] D. Xu, Q. Li, and H. Zhu, "Energy-saving computation offloading by joint data compression and resource allocation for mobile-edge computing," *IEEE Commun. Lett.*, vol. 23, no. 4, pp. 704–707, Apr. 2019.

[15] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2016, pp. 1451–1455.

[16] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3590–3605, Sep. 2016.

[17] K. Habak, M. Ammar, K. A. Harras, and E. Zegura, "Femto clouds: Leveraging mobile devices to provide cloud service at the edge," in *Proc. IEEE 8th Int. Conf. Cloud Comput.*, Jun. 2015, pp. 9–16.

[18] X. Wei, S. Wang, A. Zhou, J. Xu, S. Su, S. Kumar, and F. Yang, "MVR: An architecture for computation offloading in mobile edge computing," in *Proc. IEEE Int. Conf. Edge Comput. (EDGE)*, Jun. 2017, pp. 232–235.

[19] F. Messaoudi, A. Ksentini, and P. Bertin, "On using edge computing for computation offloading in mobile network," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2017, pp. 1–7.

[20] M. H. Ur Rehman, S. L. Chee, T. Y. Wah, A. Iqbal, and P. P. Jayaraman, "Opportunistic computation offloading in mobile edge cloud computing environments," in *Proc. 17th IEEE Int. Conf. Mobile Data Manage. (MDM)*, Jun. 2016, pp. 208–213.

[21] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.

[22] S. Ranadheera, S. Maghsudi, and E. Hossain, "Computation offloading and activation of mobile edge computing servers: A minority game," 2017, *arXiv:1710.05499*.

[23] Z. Chen, J. Hu, X. Chen, J. Hu, X. Zheng, and G. Min, "Computation offloading and task scheduling for DNN-based applications in cloud-edge computing," *IEEE Access*, vol. 8, pp. 115537–115547, 2020.

[24] Y. Sun, F. Lin, and H. Xu, "Multi-objective optimization of resource scheduling in fog computing using an improved NSGA-II," *Wireless Pers. Commun.*, vol. 102, no. 2, pp. 1369–1385, Sep. 2018.

[25] A. Enayet, M. A. Razzaque, M. M. Hassan, A. Alamri, and G. Fortino, "A mobility-aware optimal resource allocation architecture for big data task execution on mobile cloud in smart cities," *IEEE Commun. Mag.*, vol. 56, no. 2, pp. 110–117, Feb. 2018.

[26] J. Bi, H. Yuan, S. Duanmu, M. Zhou, and A. Abusorrah, "Energy-optimized partial computation offloading in mobile-edge computing with genetic simulated-annealing-based particle swarm optimization," *IEEE Internet Things J.*, vol. 8, no. 5, pp. 3774–3785, Mar. 2021.

[27] L. Kang, R.-S. Chen, W. Cao, Y.-C. Chen, and Y.-X. Hu, "Mechanism analysis of non-inertial particle swarm optimization for Internet of Things in edge computing," *Eng. Appl. Artif. Intell.*, vol. 94, Sep. 2020, Art. no. 103803.

[28] L. N. T. Huynh, Q.-V. Pham, X.-Q. Pham, T. D. T. Nguyen, M. D. Hossain, and E.-N. Huh, "Efficient computation offloading in multi-tier multi-access edge computing systems: A particle swarm optimization approach," *Appl. Sci.*, vol. 10, no. 1, p. 203, Dec. 2019.

[29] Z. Chen, J. Hu, G. Min, and X. Chen, "Effective data placement for scientific workflows in mobile edge computing using genetic particle swarm optimization," *Concurrency Comput., Pract. Exp.*, vol. 33, no. 8, p. e5413, Apr. 2021.

[30] S. Ma, S. Song, J. Zhao, L. Zhai, and F. Yang, "Joint network selection and service placement based on particle swarm optimization for multi-access edge computing," *IEEE Access*, vol. 8, pp. 160871–160881, 2020.

[31] S. Midya, A. Roy, K. Majumder, and S. Phadikar, "Multi-objective optimization technique for resource allocation and task scheduling in vehicular cloud architecture: A hybrid adaptive nature inspired approach," *J. Netw. Comput. Appl.*, vol. 103, pp. 58–84, Feb. 2018.

[32] L. Liang, J. Xiao, Z. Ren, Z. Chen, and Y. Jia, "Particle swarm based service migration scheme in the edge computing environment," *IEEE Access*, vol. 8, pp. 45596–45606, 2020.

[33] S. Azimi, C. Pahl, and M. Shirvani, "Particle swarm optimization for performance management in multi-cluster IoT edge architectures," in *Proc. 10th Int. Conf. Cloud Comput. Services Sci.*, 2020, pp. 328–337.

[34] Q. Wei, L. Liu, F. Wei, H. Ge, A. Feng, Y. Wang, and W. Li, "Computational offloading strategy based on dynamic particle swarm for multi-user mobile edge computing," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Dec. 2019, pp. 2890–2896.

[35] Y. Zhang, Y. Liu, J. Zhou, J. Sun, and K. Li, "Slow-movement particle swarm optimization algorithms for scheduling security-critical tasks in resource-limited mobile edge computing," *Future Gener. Comput. Syst.*, vol. 112, pp. 148–161, Nov. 2020.

[36] Z. Cheng, Q. Wang, Z. Li, and G. Rudolph, "Computation offloading and resource allocation for mobile edge computing," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Dec. 2019, pp. 2735–2740.

[37] A. C. Baktir, A. Ozgovde, and C. Ersoy, "How can edge computing benefit from software-defined networking: A survey, use cases, and future directions," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2359–2391, Jun. 2017.

[38] L. Dong, F. Wang, and J. Shan, "Computation offloading for mobile-edge computing with maximum flow minimum cut," in *Proc. 2nd Int. Conf. Comput. Sci. Appl. Eng.*, 2018, p. 57.

[39] M. Farshbaf and M.-R. Feizi-Derakhshi, "Multi-objective optimization of graph partitioning using genetic algorithms," in *Proc. 3rd Int. Conf. Adv. Eng. Comput. Appl. Sci.*, Oct. 2009, pp. 1–6.

[40] Q. K. Gill and K. Kaur, "A computation offloading scheme for performance enhancement of smart mobile devices for mobile cloud computing," in *Proc. Int. Conf. Next Gener. Intell. Syst. (ICNGIS)*, Sep. 2016, pp. 1–6.

[41] S. E. Mahmoodi, R. N. Uma, and K. P. Subbalakshmi, "Optimal joint scheduling and cloud offloading for mobile applications," *IEEE Trans. Cloud Comput.*, vol. 7, no. 2, pp. 301–313, Apr. 2019.

[42] D. De, *Mobile Cloud Computing: Architectures, Algorithms and Applications*. Boca Raton, FL, USA: CRC Press, 2016.

[43] P. V. Rajkumar and R. Sandhu, "Safety decidability for pre-authorization usage control with identifier attribute domains," *IEEE Trans. Dependable Secure Comput.*, vol. 17, no. 3, pp. 465–478, May 2018.

[44] R. P. V. and R. Sandhu, "POSTER: Security enhanced administrative role based access control models," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 1802–1804.

[45] G. Sharma and P. S. Rao, "Process capability improvement of an engine connecting rod machining process," *J. Ind. Eng. Int.*, vol. 9, no. 1, p. 37, Dec. 2013.

[46] Z. Li, L. Liu, and Z. Tong, "Task scheduling algorithm based on virtual machine availability awareness in cloud platform," *J. Eng. Sci. Technol. Rev.*, vol. 11, no. 5, 2018.

[47] E. S. Alkayal, N. R. Jennings, and M. F. Abulkhair, "Efficient task scheduling multi-objective particle swarm optimization in cloud computing," in *Proc. IEEE 41st Conf. Local Comput. Netw. Workshops (LCN Workshops)*, Nov. 2016, pp. 17–24.

[48] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. Int. Conf. neural Netw. (ICNN)*, vol. 4, Nov. 1995, pp. 1942–1948.

[49] M. Hristakeva and D. Shrestha, "Different approaches to solve the 0/1 knapsack problem," in *Proc. Midwest Instruct. Comput. Symp.*, 2005, pp. 1–15.

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

[51] Y. Ge and G. Wei, "GA-based task scheduler for the cloud computing systems," in *Proc. Int. Conf. Web Inf. Syst. Mining*, Oct. 2010, pp. 181–186.

[52] L. Guo, S. Zhao, S. Shen, and C. Jiang, "Task scheduling optimization in cloud computing based on heuristic algorithm," *J. Netw.*, vol. 7, no. 3, p. 547, Mar. 2012.

[53] A. M. Manasrah and H. B. Ali, "Workflow scheduling using hybrid GA-PSO algorithm in cloud computing," *Wireless Commun. Mobile Comput.*, vol. 2018, pp. 1–16, 2018.

[54] H. Li, M. Muskulus, and L. Wolters, "Modeling job arrivals in a data-intensive grid," in *Proc. Workshop Job Scheduling Strategies Parallel Process.* Berlin, Germany: Springer, 2006, pp. 210–231.

TAHA ALFAKIH received the B.S. degree in computer science from the Computer Science Department, Hadhramout University, Yemen, and the M.Sc. degree from the Department of Computer Science, King Saud University (KSU), Riyadh, Saudi Arabia, where he is currently pursuing the Ph.D. degree with the Information Systems Department. He also works as a Researcher with the Computer Science College, KSU. His research interests include machine learning, mobile edge computing, and the Internet of Things (IoT).

MOHAMMAD MEHEDI HASSAN (Senior Member, IEEE) received the Ph.D. degree in computer engineering from Kyung Hee University, Seoul, South Korea, in February 2011. He is currently a Professor with the Information Systems Department, College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia. He has authored or coauthored over 210 publications, including refereed IEEE/ACM/Springer/Elsevier journals and conference papers, books, and book chapters. His research interests include edge/cloud computing, the Internet of Things, cyber security, deep learning, artificial intelligence, body sensor networks, 5G networks, and social networks.

MUNA AL-RAZGAN received the Ph.D. degree in information technology from George Mason University, VA, USA. She is currently an Associate Professor in software engineering with the College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia. Her research interests include data mining, machine learning, artificial intelligence, educational data mining, and assistive technologies.