IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received March 20, 2022, accepted April 18, 2022, date of publication April 28, 2022, date of current version May 5, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3170918

A Cost and Energy Efficient Task Scheduling Technique to Offload Microservices Based Applications in Mobile Cloud Computing

ABID ALI[1] **AND MUHAMMAD MUNWAR IQBAL**[2]
[1]Department of Computer Science, University of Engineering and Technology, Taxila, Taxila 47080, Pakistan
[2]Department of Computer Science, Government Akhtar Nawaz Khan (Shaheed) Degree College KTS, Haripur 22620, Pakistan

Corresponding author: Abid Ali (abid.ali3@students.uettaxila.edu.pk)

This work was supported by the University of Engineering and Technology, Taxila, Pakistan.

ABSTRACT The number of smartphone users and mobile devices has increased significantly. The Mobile Cloud Applications based on cloud computing have also been increased. The mobile apps can be used in Augmented Reality, E-Transportation, 2D/3-D Games, E-Healthcare, and Education. The modern cloud-based frameworks provide such services on Virtual Machines. The existing frameworks worked well, but these suffered the problems such as overhead, resource utilization, lengthy boot-time, and cost of running Mobile Applications. This study addresses these problems by proposing a Dynamic Decision-Based Task Scheduling Technique for Microservice-based Mobile Cloud Computing Applications (MSCMCC). The MSCMCC runs delay-sensitive applications and mobility with less cost than existing approaches. The study focused on Task Scheduling problems on heterogeneous Mobile Cloud servers. We further propose Task Scheduling and Microservices based Computational Offloading (TSMCO) framework to solve the Task Scheduling in steps, such as Resource Matching, Task Sequencing, and Task Scheduling. Furthermore, the experimental results elaborate that the proposed MSCMCC and TSMCO enhance the Mobile Server Utilization. The proposed system effectively minimizes the cost of healthcare applications by 25%, augmented reality by 23%, E-Transport tasks by 21%, and 3-D games tasks by 19%, the average boot-time of microservices applications by 17%, resource utilization by 36%, and tasks arrival time by 16%.

INDEX TERMS Cloud computing, mobile cloud computing, task offloading, task sequencing, task scheduling, microservices.

I. INTRODUCTION

Mobile Cloud Computing (MCC) enhanced the tasks scheduling and task processing capabilities of Mobile Devices. Peoples use Mobile Devices to perform business tasks. MCC allows people to perform resource-intensive tasks on the cloud near mobile devices. Although tasks scheduling is part of the Mobile Devices, based on the modern application's size processing power they consume, tasks scheduling is one of the main concerns for mobile applications [1]. The mobile applications are somewhat like Augmented Reality, E-Transport, E-Healthcare, 2D/3D E-Gaming, and many more features needed to extract and process intelligent devices [2], [3]. Task Scheduling is proposed to migrate these complete tasks from resource-limited mobile devices to powerful MCC Virtual Machines (VMs). The mobile devices are now smart enough to collect, handle, and transmit the data without interruption for the concerned, intelligent devices and their effective environment [4]. This paper aims to outspread the Task Scheduling capabilities of mobile devices using Mobile Cloud Computing (MCC) [5]. However, intelligent mobile devices still face bandwidth utilization, power and battery capacity, poor CPU speed, and lower power and intensive operational capacities. Smart devices offload the computational-rich processes towards MCC for execution [6]. The goal of mobile cloud frameworks in research is to enhance the application performance and improve or save battery life for reserves restraint devices [7], [8]. These frameworks enhance the application interactivity and provide significant MCC to control the execution of these devices. According to the study, the boot time of the MCC Virtual Machines (VMs) is 28 seconds, so overheads are involved in inter-process communication among these heavy load frameworks [9]. These limitations provide us with the boost to

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

design an exceptional framework for Microservices based tasks during offloading [10].

Microservices are a collection of services for a single mobile app. This paper develops Augmented Reality, E-Transport, E-Healthcare, and 2D/3D E-Gaming applications on microservices architecture. Microservices effectively architect the application and provides frequent, rapid, and reliable application. In Mobile Devices, resource-intensive and battery-intensive applications have grown progressively in the last few years when online games, cloud-linked applications, and other resource-intensive applications are evaluated [11]. The services offered by these applications are very lightweight and oblige very tinny local services to execute them to process them correctly. The minimum delay is observed when these tinny services are offered.

On the other hand, the current oppressed heavyweight VMs provides high-level assistance for user-centric machines applications. As these services are paid according to their use model, mobility, cost, and interactivity are the main challenges to the existing MCC paradigm. Another challenge for the MCC paradigm for microservices-based applications is cost-efficient resource scheduling for Mobile processes/tasks [12]. Task Scheduling is one of the most concerning topics in mobile cloud computing due to the limited capabilities of mobile devices, storage restrictions, Task processing capabilities, and network bandwidth requirements. On the other hand, cloud-based Mobile Cloud Computing has huge processing capabilities, unlimited bandwidth, and no storage restrictions, making us use task scheduling for mobile-based microservices applications.

This paper encounters the cost-efficient task scheduling problem in MCC for IoT applications. We consider the MCC-based cloud network. The goal of the research is to curtail the cost of the services of mobile applications. We selected the computation and communication cost overhead involved in the persistent problem of task scheduling and task offloading. In Mobile applications, we have independent and fine-grained sub-tasks. Fine-grained means that every task has its attributes and data, which runs independently and effectively utilizes the workload. The associated vector attributes include every task, CPU instructions, data size, and execution deadline. We have considered the MCC services based on their price and speed.

The main contribution of the proposed system is to save time and computational energy using the mobile cloud computing approach as follows.

(i). We proposed a novel microservices container-based MCC system (MSCMCC) to implement the docker container to improve the VM's workload and enhance performance. MCSMCC gains are less overhead for services and lower boot time for VMs.

(ii). We consider MCC servers with attributes, and we also consider different Quality of Service (QoS) requirements for every task individually. We selected the MCC servers with VMs to meet the services demand. Based on the services and tasks, a service matching algorithm is proposed to compare and execute the instructions based on services requirements.

(iii). FCFC and SJF effectively sequence the task generated randomly to reduce slack time and task size.

(iv). We set up the fine-grained tasks from the local mobile device towards MCC VM to schedule the tasks. We proposed a microservices-based cost-efficient task scheduler for task scheduling to handle this problem. The proposed algorithm reduces mobile tasks' overhead cost to MCC servers.

(v). The tasks requiring special consideration during offloading are at the highest priority consideration, and sequencing is performed based on the highest priority order, a novel contribution.

The rest of the paper is organized as follows. Section 2 presents the related work on task scheduling and fault tolerance. Then, in Section 3, we outline our approach for the research and proposed solutions to the relevant problems encountered in Section 1 of the paper. Then, in Section 4, we present the proposed method with simulations using an MCSMCC approach. Section 5 concludes our proposed system, highlighting that our technique is straightforward for fault tolerance methodology.

II. LITERATURE REVIEW

With the increase of Mobile Devices and Mobile Applications, the computational offloading of mobile applications has increased and gained popularity among mobile cloud users. It allows the computationally intensive mobile devices to offload their tasks towards the cloud for mobile cloud server execution. The offloading of the tasks is performed after executing trustable tasks from the task schedular. These task offloading is improved but with limited knowledge about task microservices. [13]. The battery and mobile device performance could be improved by offloading mobile tasks. Efforts are made to perform the intensive application tasks to offload through mobile cloud support. Past studies made efforts to improve mobile application performance and save computational cost and mobile device processing power. Energy consumptions and executions time is considered in this approach for task scheduling in the provided environment [14].

In [15], the authors propose a microservices-based task offloading framework called the CEMOTS framework. The proposed technique is a mobility-aware task offloading framework that minimizes the cost of application transferring towards the cloud environment. The authors effectively reduce the application cost but do not work on VM fault rate, CPU, and resource utilization. The [16] authors proposed container-based latency and aware reliability scheduling (LRLBAS) algorithms. The Particle Swarm Optimization algorithm balances the load on edge servers and provides an effective methodology for task offloading. Another approach called MAUI performs computational offloading strategies using profiling technologies. The main difference is to take the excellent decision to offload the task either locally run or

on the mobile cloud server. The approach's primary purpose is to offload the task at its run time [17].

To the best of our knowledge, the microservices-based MCC framework for delay-sensitive fined-grained work is not proposed yet. We proposed an MCSMCC mobile cloud computing framework that executes the tasks with minimum cost and energy. Additionally, we proposed a cost-aware (CCCOF) framework to optimize the cost-centric and computational offloading framework. CCCOF provides the computational offloading with QoS, executes the services under specified deadlines, and minimizes resources costs. In [18], the authors presented another approach called ThinkAir. ThinkAir is the computational offloading framework to offload the mobile device tasks towards the mobile cloud. The idea is simple to offload the tasks when using mobile virtualization technology. Moreover, some meta-heuristic algorithmic approaches are also produced to address the task scheduling problems [19], [20].

Due to cloud network latencies, the resolution of computation offloading frameworks is not a sustainable solution. So, the cloudlet-based computational offloading resources for the wireless-based latency access network are more problem-oriented. Cloudlet frameworks brought proximity to mobile devices. In [21], Satyanarayanan *et al.* proposed a virtual machine-based Cloudlet framework. Cloudlets provide elasticity, scalability, and mobility. Cloudlets are very near to mobile phones like single hope towards the cell phones. Another researcher presented Rattrap [22] proposed an Android Cloud-based system to offload the mobile device's computational tasks to the mobile cloud by placing VMs with containers. The research aims to reduce the boot time of monolithic services for the cloud platforms using a mobile cloud-centric environment. Although the approach reduces the boot time of the services effectively, all the techniques mentioned earlier do not meet the mobile device's fine-grained requirements for resource-intensive applications.

Some research contributions contribute to and enhance the cost efficiency of mobile application services. In [23], the authors use heterogeneous mobile cloudlet services to improve cost-efficiency. The Authors consider this framework's computation time, communication time, and deadline cost. In [24]–[26], investigate, cost-efficient, and energy-efficient task offloading in mobile cloud computing are effectively probed. The focus of the research is to save the battery life of mobile devices by offloading tasks to the mobile cloud. The authors consider resources as storage. The presented computing model effectively presents the mobile cloud-based computational offloading framework for task scheduling. The [14], [27], [28] investigate the resource consumption model for mobile device applications in mobile cloud computing. Cost pricing models are addressed in these studies to provide effective collaboration for resource consumption. For example, spot instance, on-demand, and on-reserved systems are produced. The achievement of these studies is to reduce resource renting costs and execute mobile services under their deadlines.

Furthermore, the authors in [29]–[31] investigate the cost efficiency and cost-effective real-time analysis to overcome the application running cost during task scheduling. These studies aim to enhance the performance of mobile cloud applications and provide effective workflow categories to execute the tasks efficiently. Moreover, data transfer time and execution time are enhanced. Additionally, the studies considered the computational cost and communication cost of mobile device services during task scheduling in MCC [32]. In [33], the authors use mobile edge computing to minimize service latency, revenue optimization, and high quality of services. The authors share maximum revenue and services utilization features. The authors improve the utilization, service latency, revenue, and utility value but do not consider cost management, resource matching, task deadline handling, server utilization ratio, and microservices boot time. Authors in [34] design a microservice scheduling framework. The framework implements mathematics to improve the satisfaction level, network delay, energy consumption, average price, failure rate, and network throughput for the proposed technique. Still, it does not work on services delivery latency, cost management of microservices, resource matching, task failure ratio, servicer utilization, and task sequencing. Another technique presented in [35] is the author's auction mechanism to model to interact between Mobile Edge Computing systems. The performance is measured for each offloading task by the management of these tasks for justified methodology. Table 1 shows the literature in summary form. We have considered Task Sequencing, Server Utilization, Task Failure Ratio, Handling Tasks Deadlines, Boot Time, Cost Management, Resource Matching, and Services Delivery. These parameters are considered in this approach which is not considered by previous approaches as defined. The hypothesis/Baseline is illustrated in Table 1 based on the below-defined H1 to H11 hypothesis.

- **H1:** Schedule tasks based on Sequencing, Server Utilization, Fault Tolerance, Time, and Cost.
- **H2:** Task Scheduling over Sequencing, Utilization, Boot Time, and Services Delivery Latency.
- **H3:** Schedule Task Based on Sequencing, Server Utilization, Failure Ratio, Boot Time, and Services Delivery Latency.
- **H4:** Task Scheduling over Sequencing, Server Utilization, Failure Ratio, Deadline, Cost, Resources, and Services Delivery.
- **H5:** Task Scheduling using Task Sequences and Server Utilization Ratio.
- **H6:** Task scheduling for Sequencing, Network Latency, and deadline handling for tasks.
- **H7:** Task Scheduling for Sequencing, Time Management, Fault Tolerance, and Resource Matching.
- **H8:** Task Scheduling to Handle Sequencing, Fault Tolerance, and Latency values.
- **H9:** Scheduling for Task Scheduling, Deadline Handling, and Task Delivery Ratio.

TABLE 1. Comparison of related work based on characteristics with proposed system.

Papers	Hypothesis/Baseline	Sequencing	Server Utilization	Task Failure Ratio	Handling Task Deadlines	Boot Time	Cost Management	Resource Matching	Services Delivery Latency
[36]	H1	✓	✓	-	✓	✓	✓	-	-
[37]	H2	✓	✓	-	-	✓	-	-	✓
[38]	H3	✓	✓	✓	-	✓	-	-	✓
[39]	H4	✓	✓	✓	✓	-	✓	✓	✓
[40]	H5	✓	✓	-	-	-	-	-	-
[41]	H6	✓	-	-	✓	-	-	✓	-
[42]	H7	✓	-	✓	-	✓	-	✓	-
[43]	H8	✓	-	✓	-	-	-	-	✓
[44]	H9	-	-	✓	✓	-	-	-	✓
[7]	H10	-	-	✓	✓	-	✓	-	-
[45]	H11	-	✓	-	-	✓	-	-	-
Proposed Methodology	H12	✓	✓	✓	✓	✓	✓	✓	✓

- **H10:** The scheduling for tasks to get Fault Tolerance, Cost Management, and Deadline of Task to process in specified time.
- **H11:** Task Scheduling through Server Utilization, Time Management to process for task Handling.
- **H12:** The proposed approach handles the Hypothesis for Task Sequencing, Server Utilization Ratio, Task Failure Ratio, Handling Task Deadline, Boot Time, Cost Management, Fault Tolerance, Services Delivery Ratio, and Resource Matching.

The hypothesis is linked with the proposed approach to achieve more results than the state-of-the-art techniques. The results described in the proposed approach are discussed based on the hypothesis.

Most of the algorithms are proposed in this related work section. The problem we identify to address in our proposed methodology is the migration of microservices based on heavy content-sensitive applications. We address those multiple techniques proposed to address the mentioned problem but do not address the computational cost capabilities according to the requirements and demands of future needs. In the next section, we address the standard methods used to address the problem and detail about problem description.

As microservices-based tasks are growing on mobile devices, the tasks are very resource-intensive in the modern environment. So, we only need to perform the desired operation of tasks rather than whole tasks. The main reason for the microservices-based task offloading is VM-based resources and cost-effectiveness. Existing studies focus on task scheduling for microservices only on collective tasks with individual services. Still, we take both cost-effectiveness and resource handling for every individual microservices. The tasks are offloaded towards the MCC VM in this approach.

The Dynamic Decision-Based Task Scheduling Approach for Microservice-based Mobile Cloud Computing Applications framework has not been suggested to the best of our knowledge. This research proposes the MSCMCC framework for mobile devices' microservices-based tasks execution at a very low cost. Furthermore, through the MSCMCC framework, we ensure the Quality of Service (QoS) for microservices-based.

Mobile device applications over mobile cloud computing. MSCMCC framework enhances application efficiency and provides the effective resource constraints framework to enhance application-based efficiency, execute tasks under deadline, and minimize application cost.

A. PROBLEM DESCRIPTION

We study task offloading in mobile cloud computing [46], but task scheduling is an issue when mobile applications are designed through microservices containers architecture. Most current techniques focus on computational offloading frameworks to minimize the services delay, time, and cost. None of the existing techniques considers the microservices-based inter-dependent tasks and minimizes the existing cost of resource distribution, server utilization, deadline of tasks, boot time, cost management, latency, resource matching, and fault tolerance in the proposed approach. We have considered communication, offloading cost, and processing power during scheduling and offloading in CCCOF. The paper aims to reduce the computation and communication cost with processing power at MCC and Mobile Devices. In the next

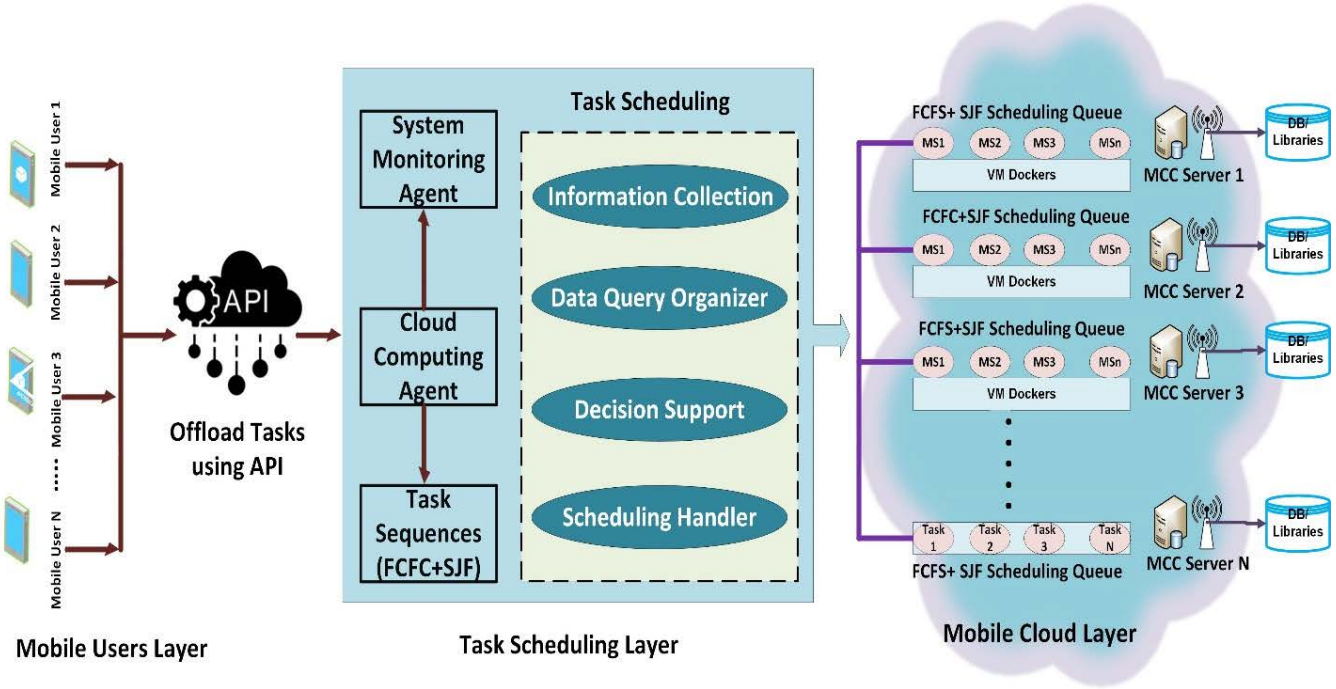


FIGURE 1. Proposed MSCMCC system architecture with three layers, i.e., Mobile users layer, task scheduling layer, and mobile cloud layer.

section of the paper, we explain the detail of the proposed architecture.

III. PROPOSED MICROSERVICES-BASED MCC ARCHITECTURE

The proposed Microservices Container-Based Mobile Cloud Computing (MSCMCC) contains Mobile Users Layer, Task Scheduling Layer, and Mobile Cloud Layer, as shown in figure 1. Generally, mobile users generate tasks for offloading. Due to their heterogeneous nature in future correspondence, these tasks are passed from the API. After passing from the API, these tasks arrived at the Task Scheduling Layer. Task Scheduling Layer consisted of four main modules. These modules receive offloaded tasks to Task Scheduling Layer. The Cloud Computing Agent (CCA) is the main module responsible and accountable for managing and handling all the offloaded tasks. The Cloud Computing Agent also assisted the System Monitoring Agent, Task Sequences, and Task Scheduling Handler. CCA is a specialized Agent which exists between mobile devices and system resources. CCA collects data from APIs of Mobile Devices, such as configuration information, metrics, and logs. All these objects that inhabit on MCC virtual network exist on MCC servers. These are System Monitoring Agent, Task Sequences, and Task Scheduling Agent. CCA utilized all these three modules for performance measurement and requested workload.

Primarily, we have used FCFC and SJF algorithms to sequence the tasks coming from Mobile devices. It is to note that, in this scheduling, every task contains vector attributes such as execution time, execution deadline, data size, and CPU instructions. Task sequences rules are proposed. Sequences rules are based on sorting algorithms to ensure the minimum cost overhead rules are occupied. System Monitoring Agent contains a table that includes tasks and their resources list. After every event in Task Scheduling Layer, the table is updated with resources consumed or left from the system or task completion. On Mobile Cloud Layer, the task is swapped from one MCC server to another, the early task scheduling is improved, and the cost function is enhanced. The task swapping among MCC servers enhances resource handling and other task provision mechanisms.

The Mobile Cloud Layer consisted of heterogeneous Mobile Cloud servers with VMs to execute and handle resource provision. All VMs resides at the top layer of MCC server layers. Internally, on every VM, the MCC Server engine adds or removes the microservices to offload the tasks. All these microservices are handled through containers. The container MCC Server Engine communicates with microservices through REST-API services. DB/Libraries are connected with the MCC servers to load any data and functions required to fulfil the Microservices offloading. All tasks are running independently and do contain their data and processes. Based on the characteristics of these tasks, every task contains a deadline for execution, data size, and CPU time to execute before scheduling from MCC. A random method loads these tasks onto the task scheduling and communication mechanism. Randomly these tasks are loaded onto the MCC system as these tasks before scheduling are not preempted on MCC.

A. TASKS MOBILITY

We attached the MCC cloud servers at the end of the MCC network. In CCA, we introduced a Mobility Module that allows the MCC network to select the mobile users or

subscribers for data packets delivery. Location Management is one of the packets delivery ratios schemes. CCA also maintains the connection of all mobile subscribers, which contains the attachment points for the final task scheduling. Handoff Management is one of the main points which decides the scenario for handoff management. The Task Mobility Section handles the glitches and functionalities of location and handoff management problems. In MCC, the Mobile Devices change their position using Mobility options, which trigger their connection from one access point to another. This process is Handoff Management. In MCC Mobile Devices connections, we divide the handoff management into three main stages. (i) Mobile Devices, MCC network agent, and MCC network changing condition causes handoff activation process of MCC Mobile Devices. (ii) Whenever a new Mobile Device connection happens, CCA performs new or additional routing and discovers a new-fangled handoff connection tool. (iii) The delivery of data from old connection points to other connection points to support the QoS operations and handling procedures. The old connection means the connection of Mobile Devices in the previous handoff process. The new connection means handoff towards the new MCC access point.

B. MICROSERVICES BASED MCC SERVERS CHARACTERIZATION OF RESOURCES

In the MCC paradigm, Mobile Cloud Layer containers are the trivial methodology. This newly thriving application methodology is especially to run the Mobile Applications in the MCC network. Classically, the administration of group containers grows in different ways, and crucial growth is observed. The deployment of the microservices containers through the MCC server layer is the fundamental problem. Microservices are tiny self-governing services inside mobile applications that communicate with external resources through well-defined APIs. Self-established teams own these microservices in MSCMCC. We consider the heterogeneous mobile cloud servers with VMs with every heterogeneous server to exploit these VMs. The MCC server engine is deployed, quickly adding, or removing the microservices. We have a heterogeneous MCC server containing devices and processors in the Mobile Cloud Layer. Every processor and device contain computational capabilities, bandwidth, runtime, costs, and VMs (container microservices).

C. RUNTIME MICROSERVICES IN MSCMCC

In microservices, the protocols are lightweight, and every service is fine-grained. In MSCMCC, every MCC server consisted of a single VM. On the other hand, every VM can support many of the microservices at a single time. Every MCC Server VM runs a single microservice according to the microservices architecture. CCA works with cost-efficient microservices with requested computational tasks to deliver efficient results. Inside CCA, every microservice (any computational task) holds libraries and resources associated with every task. Mobile Application tasks are handled through fine-grained microservices architectures during the task offloading in MSCMCC.

TABLE 2. Notations with description.

Notation	Description
N	Task set for all Mobile Applications
M	Number of MCC servers
S_{ij}	Task i is assigned to the MCC server j
y_j	Denote the ON or OFF state of the MCC server M_j
t_i	The i^{th} the task of mobile applications
M_j	The j^{th} MCC server in the model.
S_{cj}	The tasks t_i storage demand
T_d	Task t_i Deadline time.
T_w	The computational workload/data of t_i
T_{data}	Task t_i Size during transmission from Mobile device to MCC server.
B_i^j	Total bandwidth demanded by the task t_i to MCC server M_j
S_{cj}	j_{th} MCC server capacity
VM_{mic}^j	Total number of VMs deployed at MCC server M_j
ξ_j	The computational speed of individual VM under the MCC server M_j
B_{MCC}^{wj}	Bandwidth between the MCC Centric Agent and MCC server M_j during the task of offloading t_i
C_j	j_{th} fog server cost in the method

D. PROBLEM FORMULATION

Initially, we set the delay-sensitive tasks $T = \{t_1, t_2, \ldots\ldots, t_n\}$ to be offloaded to the MCC server. These tasks are scheduled through the MCC server. As we handled the microservices, so each task consisted of attributes. Attributes associated with tasks are $t_i = \{T_w, T_{data}, T_d, T_{storage}$. T_w denotes the task workload, T_{data} shows the data of task during the transmission from the mobile device to the MCC server VM, T_d illustrate the task deadline, and $T_{storage}$ Shows the task storage requirements.

Based on the MCC cloud configuration, we assume that we have N number of MCC servers, i.e., $M = \{m_1, m_2 \ldots\ldots, N\}$. Every MCC server m_j has the following attributes i.e. $m_j = \{B_{MCC}^{wj}, \xi_j, S_{cj}, VM_{mic}^j\}$, where B_{MCC}^{wj} is the bandwidth between the MCC Centric Agent and MCC server during the task offloading, ξ_j demonstrates the computing rates of J_{th} MCC server, S_{cj} Demonstrates the total storage of the MCC server j in the method. VM_{mic}^j Demonstrates the deployed VMs for microservices through MCC server j with the same capability to handle tasks. Each VM_{mic}^j It comprises multiple containers to execute the microservices to execute multiple tasks. Each microservice has its database and libraries to be executed during the execution.

When a task is scheduled at MCC server j, we illustrate that B_i^j does the task demand the total bandwidth t_i. The resources

such as Storage, RAM, Bandwidth, etc., t_i is the offloaded task towards MCC server cost, i.e., j, and the state of the task is $y_j = 1$ are denoted by C_j. Due to limited page space, this paper shows a limited explanation of the notation, and the remaining notations are illustrated in Table 2. Equation 1 illustrates that through binary variable s_{ij} either the task t_i scheduling through the MCC server M_j Or not. s_{ij} can be either zero or one for tasks scheduling overall tasks from all applications. The decision about tasks is made through the proposed servers.

$$S_{ij} = \begin{cases} 1, & t_i \leftarrow M_i \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

Moreover, each task in this problem context is assigned to one M_j. On the other hand, the MCC server schedules one task simultaneously. Equation 2 shows the task assignment t_i to MCC server M. The one indicates the ready tasks assigned to the MCC server for processing on its VMs. The value 1 indicates the positive task assignment to the appropriate server VM from 1 to N tasks.

$$\sum_{j=1}^{N} S_{ij} = 1 \tag{2}$$

Every MCC server has limited capability regarding resources. Therefore, the task offloading requirements do not exceed the MCC server capabilities. Equation 3 represents the scenario to check the task offloading capabilities about tasks scheduling. Task offloading capabilities are checked for s_{cj}, s_{ij} against s_{cj} to check that no tasks processing requirements exceed the MCC server.

$$\sum_{i=1}^{N} S_{cj} * S_{ij} \leq S_{cj} \tag{3}$$

The MCC resource server has the capabilities of limited resources. It also has limited VM to offer its services to execute the microservice for each task coordination. So, equation 4 represents fewer computational tasks assignment decision making. Every requested task (microservices tasks) must consume less computational than VM computational capabilities.

$$\sum_{i=1}^{N} T_w * S_{ij} \leq VM_{mic}^{j} \tag{4}$$

Every task is scheduled on an optimal MCC server M_j. Cloud Computing Agent decides where to schedule the designated task t_i Towards the MCC server VM. The task execution on the MCC server M_j is illustrated through equation 5. It illustrates that task with resources, storage, and time responsible for selecting the appropriate server M_j to follow for final decision.

$$T_{t_i}^{e} = \sum_{j=1}^{M} \frac{T_w}{R_{M_j}} * S_{ij} \tag{5}$$

Therefore, after the decision of the Cloud Computing Agent, when a task t_i It is scheduled towards the MCC server for computation; the task gains extra computational offloading possibilities and sends the results back to the Mobile Device from the MCC server. The computational work is illustrated through equation 6.

$$R_T^T = \left(\frac{T - Date_i^{enter}}{B_{MCC_{ij}}^{wj\ up}} + \frac{T - Date_i^{leave}}{B_{MCC_{ij}}^{wj\ down}} \right) \tag{6}$$

In equation 6 the $T - Date_i^{enter}$ illustrate the task t_i input data size and $T - Date_i^{leave}$ shows the task t_i the output data size after being processed by the MCC server M_j. $B_{MCC_{ij}}^{wj\ up}$ and $B_{MCC_{ij}}^{wj\ down}$ Shows bandwidth of uplink and downlink of link rates from Mobile device offloading to MCC server and gets results back to the mobile device after computation. R_T^T is the Round-Trip Time, i.e., the time between data sending and receiving from Mobile Device to MCC server M_j For all tasks (microservices). Now we measured the bandwidth required for the measurement of each task. Equation 7 illustrates the bandwidth requirements for every task for R_T^T. The bandwidth requirements are checked against each task to best fit MCC server VM processing capabilities and data communication with tasks.

$$S_{ij} \left(R_T^T + T_{t_i}^{e} \right) \leq T_d \tag{7}$$

The above equation 7 defines the bandwidth requirements, but as every task is different, the bandwidth differs. We obtained inequality bandwidth of task t_i is illustrated in equation 8. The bandwidth is checked against the Time, Data, The computational speed of individual VM under the MCC server M_j. The inequality bandwidth is measured through equation 8.

$$B_{MCC}^{wij} > \frac{T - data_i}{T_d - \frac{T_w^i}{\xi_j}} \tag{8}$$

To achieve the performance of the proposed system, we make sure that all the tasks t_i Originated from Mobile Devices should be finished before their respective deadlines. This process minimizes the MCC Server M_j Cost. The required communication bandwidth between the MCC server and Mobile application is calculated in equation 9. After inequality bandwidth of task t_i. The communication bandwidth is computed to check the actual and expected communication bandwidth required for the task to offload.

$$B_{MCC}^{wij} = \frac{T - data_i}{T_d - \frac{T_w^i}{\xi_j}} \tag{9}$$

Besides the individual task bandwidth, every MCC server M_j has limited bandwidth with limited VMs. The allocation of the tasks to the MCC server M_j it should contain less bandwidth consumed as compared to the server itself. Equation 10 illustrates total bandwidth of the tasks should be less than or equal to the bandwidth required to MCC for wj tasks.

wij shows the bandwidth required to execute the tasks for proper ordering.

$$\sum_{i=1}^{N} B_{MCC}^{wij} * S_{ij} \leq B_{MCC}^{wj} \tag{10}$$

Cloud Computing Agent (CCA) is an instigator that connects and monitors the MCC server. CCA monitors its performance during every time interval. MCC Server's cost depends on two main elements: State and resources required for microservices for every offloaded task on the MCC server. User Mobile application tasks are handled through CCA. Its cost is not only dependent on its On State, but it only charges for the requested computational capabilities of the MCC server. To show the status of the MCC Server, we adopted a binary variable ∂_j To show state through equation 11. ∂_j shows the on and off condition of the server for tasks processing.

$$\partial_j = \begin{cases} 1, & M_j \leftarrow \text{on} \\ 0, & \text{off}, \end{cases} \tag{11}$$

E. MCC COST MODEL

Microservices are not individual computational applications. These included the cost model, which explains the on-demand resource access method, ensuring connectivity based on the business applications framework. Equation 12 illustrate the on-demand cost model for mobile application (Typically the business applications). This model computed the processing demand for every selected application used in the simulation.

$$C_j = \delta_j * S_{ij} * T_{t_i}^e \tag{12}$$

In table 2, we show the δ_j Values as unit price for computational work for every individual MCC server. Now it is time to compute the resource constraints for every MCC server. The formulated constraints are computed through equations 13 to 25. These equations collectively compute the resources constrained for MCC servers MCC_1 to MCC_3. The state of every server is monitored through ∂_j CPU costs, bandwidth requirements, and microservices-based processing tasks must be distributed against every resource. Finally, the $S_{ij} = \{0, 1\}$ decides about the task offloading using selected parameters in table 3. Equation 13 computes the *min* R_c for every task for the resources demand. *min* R_c depended on On-Demand $\boldsymbol{\delta_j}$ and MCC Server State ∂_j. Equation 14 compute *min* R_c for the task to compute. Equation 16 computes the time for resources to compute every task on j server. Similarly, equation 17 computes the i^{th} the task of mobile applications for task tasks from table 3. Equation 18 and 19 computes mobile Cloud-based resources tasks comparison and execution and resources required to compete against each task. Moreover, equation 20 and 21 computes that j_{th} MCC server capacity for resources and set it equal to 1, which shows that resources are convinced to offload the task towards MCC servers. Equations 22, 23, and 24 decide about server, bandwidth, and VM-based resources required for every microservices-based task to offload towards the cloud. The final decision is computed in $S_{ij} = \{0, 1\}$ to get the task for offloading with required resources.

$$\min R_c = \sum_{i=1}^{M} \sum_{j=1}^{N} \partial_j * C_j \quad \forall\, i \in N \tag{13}$$

$$\text{subject to min } R_c = \sum_{i=1}^{M} \sum_{j=1}^{N} S_{ij} * \partial_j * C_j \quad \forall\, i \in N \tag{14}$$

$$T_0^j = 0, \quad \forall\, \{j = 1, 2, \ldots\ldots.., N\} \tag{15}$$

$$T_k^j = T_k^j - 1 * \sum_{k=1}^{N} S_k^j * T_{t_k}^e \quad \forall\, \{j = 1, 2 \ldots\ldots., N\} \tag{16}$$

$$T_{t_i}^e = \sum_{j=1}^{N} S_{ij} \times \frac{T_w}{\xi_j} \quad \forall\, \{i = 1, 2 \ldots\ldots., N\} \tag{17}$$

$$MC_i = \sum_{j=1}^{N} T_k^j * S_{ij} \quad \forall\, \{i = 1, 2 \ldots\ldots., N\} \tag{18}$$

$$MC_i + R_T^T \leq T_d^i \tag{19}$$

$$\sum_{j=1}^{N} S_{ij} = 1, \quad \forall\, \{i = 1, 2 \ldots\ldots., M\} \tag{20}$$

$$\sum_{j=1}^{M} S_{ij} = 1, \quad \forall\, \{i = 1, 2 \ldots\ldots., N\} \tag{21}$$

$$\sum_{i=1}^{M} S_{ij} * S_{ci} \leq S_{cj}, \quad \forall\, j \in 1, 2, \ldots..N \tag{22}$$

$$\sum_{i=1}^{M} S_{ij} \leq VM_{mic}^j, \quad \forall\, j \in 1, 2, \ldots..N, \tag{23}$$

$$\sum_{i=1}^{M} S_{ij} * B_i^j \leq B_{MCC}^{wj}, \quad \forall\, j \in 1, 2, \ldots..N, \tag{24}$$

$$S_{ij} = \{0, 1\} \tag{25}$$

F. PROPOSED ALGORITHM FRAMEWORK FOR SYSTEM

Task Scheduling and Microservices based Computational Offloading (TSMCO) framework comprises several components. Figure 2 shows the TSMCO framework components. The first module of the framework is the resource matching from incoming tasks from mobile devices and then matching for MCC servers for their pair-wise processing. Task Sequencing is one of the most critical components of

TABLE 3. MCC servers unit price.

MCC Servers	On-Demand δ_j			MCC Server State
	B^{wj}_{MCC} /Cost	S_{cj}/Cost	C_{CPU}/Cost	∂_j
MC_1	0.4	0.2	2	0/1
MC_2	0.6	0.4	4	0/1
MC_3	0.8	0.6	6	0/1

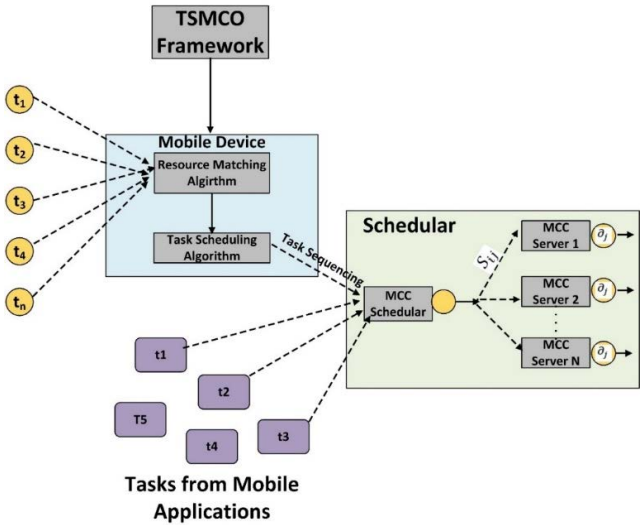


FIGURE 2. TSMCO framework to get microservices-based tasks from mobile devices with resource matching algorithm and task scheduling algorithm is implemented to offload task, and MCC Schedular uses VMs to process the offloaded task.

the TSMCO framework. It uses sorting algorithms to sort the tasks into different to perform cost-effective scheduling. Task sequencing provides input to the tasks scheduled. The task sequence t_i to be scheduled on the MCC server MC_j if $S_{ij} = 1$, or else $S_{ij} = 0$. The task sequences process continues until all the tasks are scheduled and processed according to their deadlines using MSCMCC. All the mobile tasks are passed and processed from multiple components and complete their executions. We have heterogeneous MCC servers deployed to process the scheduled tasks on each VM. We define Algorithm 1 to best schedule for tasks. Task Scheduling Mobile Cloud Computing Optimization (TSMCO) algorithm is declared for task scheduling on MCC servers. This algorithm takes G, T, and j/M as input, and output returns resource constraints for every MCC server. Initially, the decision schemes are called with server state ∂_j, cost C_j, and Services S_{ij}. In steps 2 and 3, tasks scheduling and sequencing are performed using said parameters, i.e., G, T, M. The array of tasks from mobile nodes is defined on the empty step. Initially, it was set to NULL with no entries. All the processes continue for M, G, and T on steps 5, 6, and 7. Steps 8 and 9 define checking for time. If defined, then time for CPU, data, execution of tasks, and limit for tasks execution is declared. This checks resources also to be identified for provided time. Next, in steps 10 and 11, the tasks list is checked. If not empty, then sequencing of ready tasks is performed for M, T, and G parameters. Step 12 shows an array of tasks ready for offloading decision. Then, in steps 13 to 15, the resource cost R_c is computed for every task found in the array of ready tasks and returned to the system for further actions.

Algorithm 1 : TSMCO

Input: $G = \{g_1, g_2, \ldots., g_n\}$, $T = \{t_1, t_2, \ldots\ldots, t_n\}$, $j \in M = \{m_1, m_2 \ldots\ldots, N\}$.
Output: $\min R_c$
Steps
1. $\sum_{i=1}^{M} \sum_{j=1}^{N} S_{ij} * \partial_j * C_j$ Call (Decision Scheme)
2. task_schedule(G, T, M)
3. task_sequence(M, T, G)
4. $T^{Mobile}_{LIST}[] \leftarrow$ NULL
5. while(M) do
6. while(G) do
7. while(T)
8. if ($T \neq \varphi$)then
9. resource_algo(T^{time}_{exe}, T^{limit}_{exe}, T^{req}_{data}, T^{ins}_{cpu})
10. if($T^{Mobile}_{LIST}[] \neq \varphi$)then
11. task_sequence(M, T, G)
12. *task_sequence*(M, T, G)
13. $R_c* \leftarrow (\sum_{i=1}^{M} \sum_{j=1}^{N} \partial_j * C_j) * \leftarrow T^{Mobile}_{LIST}[]$
14. $R_c \leftarrow R_c*$
15. Return R_c;
16. End Loop;

G. MCC SERVER RESOURCE ATTAINING

We deal with the cost optimization problem [47] and heterogeneous MCC servers. The most effective and interesting is selecting the best edge MCC server to process all the extended sequential tasks. The object of our task scheduling for microservices applications is to reduce MCC's processing and computation costs. The MCC server selection with minimum cost δ_j Is one of the challenging ways. Equation 26 and equation 27 illustrates the unit cost δ_j And smaller MCC server costs, respectively.

$$\delta_j = \frac{C_j}{\rho_j} \tag{26}$$

In equation 26 the ρ_j define the size of the MCC server M_j. The cost of the server in respect of processing power, memory, and tasks processing size. The output of equation 26 is the total cost from the selected MCC server. The unit cost δ_j is determined through equation 27. The unit cost is determined through MCC tasks demand, MCC VM demand, and MCC bandwidth handling demand. This originated for

tasks to be offloaded with computed cost.

$$\rho_{\mathrm{j}} = \frac{\mathrm{MCC}_{\mathrm{S_{cj}}}}{\sum_{\mathrm{i=1}}^{\mathrm{N}} \mathrm{S_{cj}}} + \frac{\mathrm{MCC}_{\mathrm{VM_{mic}^{j}}}}{\sum_{\mathrm{i=1}}^{\mathrm{N}} \mathrm{VM_{mic}^{j}}} + \frac{\mathrm{MCC}_{\mathrm{B_{MCC}^{wj}}}}{\sum_{\mathrm{i=1}}^{\mathrm{N}} \mathrm{B_{MCC}^{wj}}} \tag{27}$$

After successful processing from the servers, the remaining resourcing is not needed to waste. $\mu_{\bar{M}_i}$ are considered as remaining resources on the MCC server M_i After initial level task scheduling. There is a task that maximizes the products and their primary operations, so we denoted it dot product γ_i. The determined values of γ_i is computed through equations 28, 29, 30, and 31. These equations show complete declarations for the proposed system.

$$\gamma_{\mathrm{i}} = \overline{\mu_{M_i}} * \overline{\beta_{\tau_{ij}}} \tag{28}$$

$$\gamma_{\mathrm{i}} = \mathrm{S_{ci}} * \mathrm{R_j^l} * \mathrm{q_i} + \mathrm{q_i} * \nu_{\mathrm{j}}^{\mathrm{l}} + \mathrm{c} * \mathrm{b_j^l} * \mathrm{q_i} \tag{29}$$

$$\mu_{\mathrm{M_i}} = \left(\mathrm{R_j^l} * \mathrm{q_i} * \nu_{\mathrm{j}}^{\mathrm{l}} * \mathrm{q_i}, \mathrm{b_j^l} * \mathrm{q_i}\right) \tag{30}$$

$$\mu_{\mathrm{M_i}} = \mathrm{Y_{M_j}} - \sum \beta_{\mathrm{t}_{*j}} \tag{31}$$

$\beta_{t_{*j}}$ illustrate the tasks resource management to schedule on the MCC server M_j. There are different types and resources on the MCC server system. Resource matching is used to allocate the optimal and best available resource for each task in the MCC server in heterogeneous nature. The tasks contain vector attributes such as task deadline, data size, and workload. On the other hand, the resources contain vector attributes such as bandwidth, cost, VM capacity, and storage. Resource matching is one of the problems that must be addressed. We select Techniques for Order of Preference by Similarity to Ideal Solution (TOSS) model and Analytic hierarchy processing (AHP) model as resource matching algorithms. Algorithm 2 is designed for MCC server resources matching. It takes resources and tasks (in sequential order) as input. The output of the algorithm is the *FLIST* [] frequent list array. *FLIST* [] after fulfilment of the requirements of the task linked with resources, store these on the MCC server MC_j. The detailed explanation of algorithm 2 is as follows.

In algorithm 2, step 4 verified the needs of the MCC server for all incoming tasks $\mathrm{t_i}$ If the resources are matched according to the MCC server, the algorithm returns true, otherwise returns false. After the return of true results, we add the matching list to Frequent List $PLIST[k_j, t_i]$. Step 4 repeats all the possible tasks coming from mobile devices to match the heterogeneous MCC server requirements.

Algorithm 2 : Resource Matching on MCC Server

Input: $\mathrm{G} = \{\mathrm{g_1}, \mathrm{g_2}, \ldots, \mathrm{g_n}\}$, $T = \{\mathrm{t_1}, \mathrm{t_2}, \ldots\ldots, \mathrm{t_n}\}$, $\mathrm{t_i} \leftarrow \{\mathrm{S_{cj}}, \mathrm{T_w}, \mathrm{T_d}\}$, $\mathrm{k_j} \leftarrow \setminus \{\mathrm{R_j^l}, \nu_{\mathrm{j}}^{\mathrm{l}}, \nu_{\mathrm{j}}^{\mathrm{l}}\}$
Output: FLIST[]
Steps
1. Start
2. while(G) do
3. while(T) do
4. if($\mathrm{t_i} \equiv \mathrm{k_j}$)then
5. AHP($\mathrm{R_j^l}$, $\nu_{\mathrm{j}}^{\mathrm{l}}$, $\nu_{\mathrm{j}}^{\mathrm{l}}$);
6. TOSS($\mathrm{S_{cj}}$, $\mathrm{T_w}$, $\mathrm{T_d}$);
7. add(PLIST[$\mathrm{k_j}$, $\mathrm{t_i}$]);
8. else
9. gotoStep4;
10. *store*(PLIST($\mathrm{k_j}$, $\mathrm{t_i}$);
11. Return PLIST[$\mathrm{k_j}$, $\mathrm{t_i}$];
12. End Loop;
13. End;

H. MCC SERVER TASK SEQUENCING

All tasks arrived randomly because mobile devices originated from different devices. The arrival rate of the tasks is followed through Poisson Process. The task sequences do not follow any rule to allow the task to allocate exclusive of any sequence randomly. All the arrived tasks must be sequenced first to be distinguished from non-sequenced and sequenced tasks. The sequenced must be in proper format and provided in proper sequential order. The company task sequenced method consists of four rule-based. The method is deployed in the current system. We take task size, deadline, and slack time as three parameters to sequence the tasks. Four rules are developed and deployed to sort all incoming un-ordered tasks. The rules are as under: -

1) First Come, First Served (FCFS): We sort the task according to their arrival time. All the tasks sort according to their incoming time to queue. No such priority is maintained due to its FCFS nature. Late tasks are sorted according to their incoming time. The tasks lateness through FCFS is accessed according to the effective way [34].

$$\mathrm{FCFS} = \mathrm{T_d} - \mathrm{M_i} \tag{32}$$

1) Shortest Job First (SJF): The tasks are sorted according to their computational time. The shortest lateness tasks are scheduled first, and lengthy executed tasks are executed later.
2) Shortest Size First (SSF): In this strategy, all the tasks are sorted according to the size of the task. Shorter size tasks are arranged first, and lengthy size tasks are arranged later. The SSF follows the strategy of pre-emptive task scheduling for sequences of the tasks, which is very helpful for the efficient and reliable provision of the data.

All the offloaded tasks from mobile devices randomly arrived at MCC. Initially, FCFS arranged all these tasks in a first-come, first-severed sequence. FCFS executes the order sequences rules to arrange the tasks in a particular order in a cost-efficient method. Figure 3 exploits all task sequences techniques, from task offloading to task sequencing. Every task sequence method, i.e., FCFS, SJF, and SSF, has different scheduling and sequences results. However, we will select one with the best results according to the objectives of the problem for optimal task sequencing.

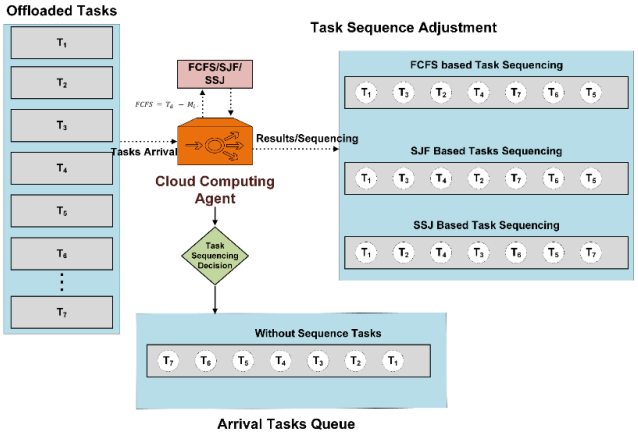


FIGURE 3. Tasks sequences adjustment for applications based on arrival time using cloud computing agent using FCFS, SJF, and SSJ task schedular.

I. TASK SCHEDULING

Till now, we have done the resource matching and tasks sequencing. After that, we should get the tasks scheduling methodology adopted to sort the problem. However, the initial implementation of the tasks scheduling is not just the outcome of the task scheduling policies implemented in the paper. The cost can be calculated in some different ways. Due to concurrent changes in the MCC cloud network, the initial selection is not suitable for task scheduling to calculate the Mobile application cost. Another reason not to select the initial as final is instability in the MCC resources. A new, improved solution is required to improve task scheduling. For example, we take tasks, i.e., T_1, *and* T_2 to execute on heterogeneous MCC server with VM i.e. K_1, *and* K_2. The resources required for T_1, *and* T_2 are (deadline: 24, data size: 15Mb, and CPU required: 12) and (deadline: 50, data size: 35Mb, and CPU required: 32) respectively. While the resources attribute to MCC Servers K_1, *and* K_2 are (R_j^l : 15, v_j^l : 4, b_j^l : 10) and (R_j^l : 20, v_j^l : 8, b_j^l : 20) respectively.

Initially, the task T_1 is scheduled on the MCC server K_1, and task T_2 is scheduled on the MCC server K_2. The total cost of the application is the total aggregate of the two MCC server costs. Equation 33 depicts the total cost of the application. The equation effectively shows the reduced cost required for the application to offload the task.

$$\omega_{\text{total}} = \sum_{\text{i=1}}^{2} \text{K}_\text{i} * \text{T}_\text{i} \tag{33}$$

If all the tasks are scheduled on the MCC server K_2 then the total cost of application is solely the cost of the MCC server K_2. Although the tasks are executed on the same server, this reduces applications computational costs. Figure 4 shows more than one solution for single task sequencing. We are also ready to accept the worst solution to be adopted by the processes. The challenge for the schedular is to pick one of the best solutions. The schedular should pick one solution which reduces the system's total internal cost. In the example mentioned above, the MCC server picked by the Schedular has resources according to the cost mentioned in the resource

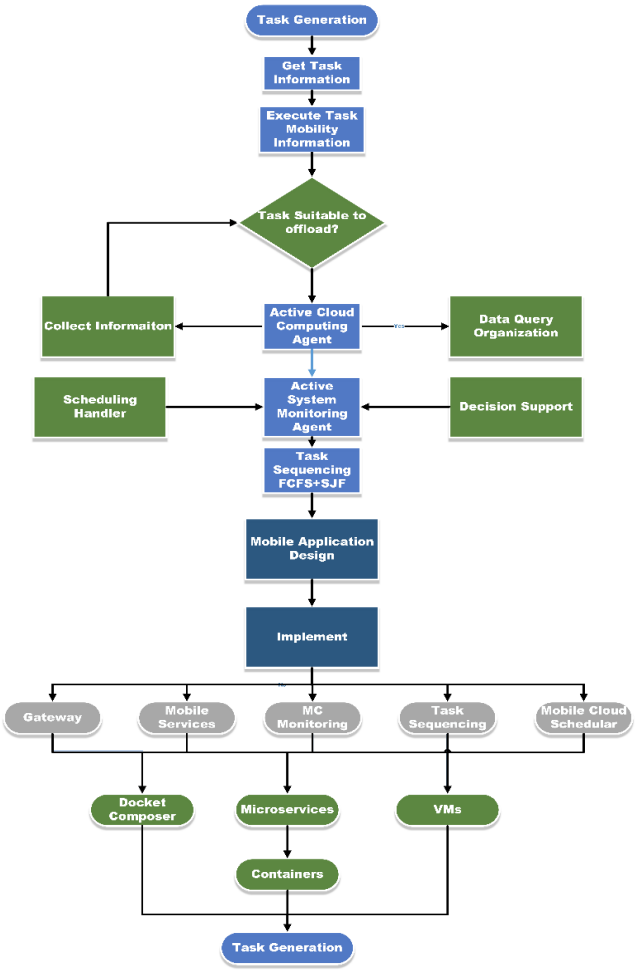


FIGURE 4. Flow chart of the complete working of the proposed system with implementation detail of simulation work.

list. So, additional optimization is required to achieve high-cost reduction. At the start, the MCC Server had a high cost of available resources. The challenging thing for us is how to reduce the resource utilization cost and utilize the maximum resources. So, we introduce the improved Task Scheduling methodology that significantly improves the resource utilization of MCC servers. The main goal of the scheduling is to schedule the tasks to MCC servers with the lowest cost at this initial scheduling phase. This is one only way through which the Task Scheduling Algorithm improves the performance of the MCC server. The scheduler removed the extra and most expensive cost at this initial collaboration stage. However, we propose task scheduling algorithms that enhance the scheduling performance and solve the resource optimization problem on MCC Server. Algorithm 3 defines the task scheduling on the MCC server with optimized resource utilization. The algorithm takes input tasks set to be scheduled on a heterogeneous MCC server. Algorithm 3 is executed in the below-mentioned task as:

J. TIME COMPLEXITY OF TSMCO

The TSMCO has different components like Resource Matching, Task Sequences, and Task Scheduling. We take these

Algorithm 3 : Task Scheduling Algorithm

Input: $G = \{g_1, g_2, \ldots., g_n\}$, $T = \{t_1, t_2, \ldots\ldots, t_n\}$, $M_i \leftarrow \setminus \{M_1, M_2, M_3, \ldots.., M_n\}$
Output: S_{ij}(TaskScheduling), γ_j(MCCServerState)
Steps

1. $S_{ij} \leftarrow 0$; : *Binary Variable Declaration to zero*
2. Initialize $\leftarrow \min R_c$;
3. $\gamma_j \leftarrow 0$; : *MCC Server state declaration* to zero
4. VectorAttribute $\rightarrow Y_{M_j}$;
5. Unit Cost δ_j of MCC servers $K \in M_i$;
6. $\omega_{total} = \sum_{i=1}^{2} K_i * T_i$: *Determine the total cost*
7. $J_j = \{\}$; : *Exploited MCC Server* Set
8. begin
9. while(G)do
10. *While(T) do*
11. While(M_i)*do*
12. if($(M_i \leftarrow G) \neq \emptyset$) then
13. Resource match $\rightarrow \delta_j$ (smallest MCC cost)
14. $t_i \leftarrow M_j$;
15. $h_t^R \equiv M_j$;
16. t_i(largestsize) $\rightarrow M_j$
17. $T_i \leftarrow T_i \,|\{t_i\}|$;
18. Establish $\{S_{ij}, \gamma_j\} = 1$;
19. $J_j \leftarrow J_j \cup \{M_j\}$;
20. $minR_c \leftarrow S_{ij}$: *Optimal Task Assignment*
21. $M_j = M_j \cup \{M_j\}$;
22. $\{M_{gi}, M_{g2}\} \leftarrow W$;
23. if($(M_{g1} > M_{g2}) \leftarrow E$)then
24. swap($t_i(M_{g1})tot_i(M_{g2})$)
25. *assign*$(S_{i(g1)} = 1)$
26. $(minR_c)* \leftarrow S_{ij}$: *Optimal Assignment*
27. *elseif* $(t_i \leftarrow M_{g2} = \emptyset)$ *then*
28. *set*$W \leftarrow W \,|M_{g2}|$
29. *set*$\gamma_{f(g2)} = 0$; : *New Fog server*
30. M_{g2}(*smaller Cost E*);
31. *End while*;
32. *End While*;
33. *End* $-$ *Loops*;

three components separately and then effectively utilize all these components to obtain the time complexity. (1) Resource Matching: We expect the heterogeneous servers to exploit the TOPSIS and AHP methods for task matching [44]. The calculated time complexity of the Resource Matching is $O(MxT)$ [45]. M is the MCC server resources for multi-criteria, and T is the tasks arranged in the pair-wise matching. (2) Sequence of Tasks: In Task sequencing, all the tasks are sorted with the shortest size, deadlines, and lateness by using $O(mlogn)$. N is the number of sorted tasks, and M is the exploited method to sort the tasks accordingly. (3) Task Scheduling: All the MCC servers are scheduled according to descending order of δ_j, and C_j. Although the time complexity that we have measured is $O(logM).O(logM) + N$ [46]. This time complexity is for all MCC servers according to their descending order of price and load for the task scheduling process. N shows the tasks swapping process in the time complexity of the different MCC servers. Figure 4 represents the entire system work, including the simulation work.

IV. PERFORMANCE EVALUATION

To evaluate the performance of proposed TSMCO and MSCMCC frameworks, we generate practical results from different simulators over mobile and microservices-based applications. Initially, we simulated task sequencing over FCFS, SSJ, and SJF to simulate the task sequencing results of the proposed methodology. Task sequencing is an evaluation metric that defines the task's actual sequencing before the final decision. Furthermore, the CPU utilization evaluation metric is defined to check existing resource utilization under the common ship of different resources. Overhead time is an evaluation metric. Overhead time describes the framework for microservices overhead time using performance parameters.

Task Deadlines (Execute all the tasks before the final submission of the deadlines), Cost (RPD value in percentage to show the total cost of applications such as E-Transport, 2D/3D Games, Augmented Reality, HD Video Streaming, and Healthcare Applications used in the proposed system evaluation), and task failure ratio (During Scheduling Task failed to offload, and VMs failure after offloading) evaluation metrics are evaluated in the results section to briefly describe the performance of the proposed microservices-based task offloading framework. Table 4 defines the stimulation parameters with their description.

The process is divided into distinct parts based on the simulation parameters defined in Table 4. (1) MSCMCC implementation part, (2) Metric Parameters and Components Calibration, (3) Comparison of TSMCO offloading framework, and (4) Algorithms Comparison and Task Scheduling part. In Table 5, we describe the MCC server's resources, and in table 6, we describe the workload analysis of Mobile Applications. So, selecting these applications is to compare with existing techniques from literature to get better results for this cost-intensive application. Every application instance is considered one task containing multiple microservices.

A. COMPARISON FRAMEWORK AND APPROACHES

To compare with the existing approach, the following primary considerations are considered to compare the obtained results. The hypothesis means what we are trying to predict from the baseline of the proposed outcomes. In the proposed framework, the hypothesis defines the proposed approach with compared results with the baseline technique.

- **Hypothesis-1/Baseline-1:** We implement the VM-based offloading framework for MCC task scheduling. The studies have implemented the [46], [43], [48], [49] frameworks for testing results. The goal of the hypothesis is to offload the entire mobile application (including all the microservices) to the MCC cloud servers.
- **Hypothesis-2/Baseline-2:** Dynamic computational offloading-based framework is implemented using

TABLE 4. Simulation parameters.

S.No.	Parameters	Used Values
1	Windows OS	Docker Engine
2	Language	Python, JAVA, XML
3	Processor	X64 bit
4	Android Phone	LG W11, W31
5	Simulation Time	14 Hours
6	Simulation Repetition	180 Times
7	Android OS	Orio
8	App Interface	Android Orio Interface
9	Simulation Evaluation	Single and Multiple Factor Anova
10	Service	EC2 (Amazon on Demand)
11	$B_{MCC}^{wj(up)}$	50 ≈ 1200 Mbps
12	$B_{MCC}^{wj(down)}$	1200 ≈ 50 Mbps

TABLE 5. MCC server specifications.

Cloud Name	VM Core	MIPS/Core	VMs	Storage	Cost
j_1	i3	1000	2	800	0.6
j_2	i5	3000	4	2600	0.7
j_3	i7	5000	8	4000	0.9
j_4	i9	10000	14	10000	0.05

TABLE 6. Mobile device applications workload analysis.

Applications Workload	$T_{w,i}$(MB)	*Communication Cost*	*N*
E − Transport	40.2	4G:0.7$	650
2D/3D Games	31.2	3G:2.5$	790
Augmented Reality	25.1	4G:1.5$	650
HD Video Streaming	52.8	5G:5$	900
Healthcare Applications	16.5	5G:2.6$	830

virtual machines. The adopted strategies are [50]–[52] used to test and compare the testing results. The aim is to offload complete mobile applications towards heterogeneous servers. The offloading is based on sufficient available resources.

- **Hypothesis-3/Baseline-3:** Mobile Applications deadlines are implemented in the baseline for Cost and Application deadline time. These are five selected application types taken from table 5 [16]. These are used to test and compare the testing results in Figures 9 and 10.

These methods work for cost, energy, and fault rate handling using task dependency values, Cloud Assisted Mobile Edge Computing (CAME) framework in which cloud resources are leased to enhance the system computing capacity. Another baseline method called Rattraap is used to compare the proposed system results. A lightweight cloud platform that improves the offloading performance from the cloud side. The results are compared for boot time for microservices, overhead cost, tasks sequences, deadline for microservices, failure rate, and tasks rates for microservices. These results are compared with the baseline methods for effective results comparison. Initially, we build the model for the proposed system and compare the selected results with existing baseline methods. We review that our results are comprehensively better than these baseline methods from all aspects.

B. PERFORMANCE METRICS

In this paper, the components collaborations recommend the actual implementation plan for experimental results. The application tasks are generated from components collaboration as described in table 3. In this experiment, we take five different applications, and their types are considered for experimental results. Every task contains a deadline to finish its execution. The task deadlines set in this experimental paper setup are based on equation 34. This equation computes the task deadline for task requirements and provides a comprehensive approach to dealing with the finished time of tasks.

$$\mathrm{T_d^{a,i}} = \mathrm{P_{a,i}} + \gamma + \mathrm{P_{a,i}} \tag{34}$$

The task deadline for $T_d^{a,i}$ is required to define the early finished time and task deadline to finish the final task interval. γ shows the tightness in the task deadline, which shows the values of 0.2, 0.4, 0.6, 0.8, and 1. So every task on a mobile device contains five different deadlines for tasks, i.e., *D1*, *D2*, *D3*, *D4*, and *D5*. Using equation 31, we verify the algorithm performance throughout the performance metrics. Relative Percentage Division (RPD) statistical analyses are performed to compute the recital division method for MTOP, VFCN, and CTOS. The paper results effectively evaluate the power consumed by the different devices to effectively participate and provide the algorithm to evaluate the computational throughput for effective parameters. Equation 34 defines the RPD estimation for the proposed technique. This equation experience provides a percentage of task offloading.

$$\mathrm{RPD}(\%) = \frac{\mathrm{P_a^*} + \mathrm{P_a}}{\mathrm{P_a^*}} \times 100\% \tag{35}$$

where, P_a shows the objective function.

C. MSCMCC IMPLEMENTATION

We implement a Mobile Cloud-based application in Mobile Devices using Mobile Application Developer IDE, i.e., Android Studio, and Huawei Y9 2019 Mobile Model as an emulator to test mobile applications. Figure 5 shows this description for applications and scenarios. We evaluate

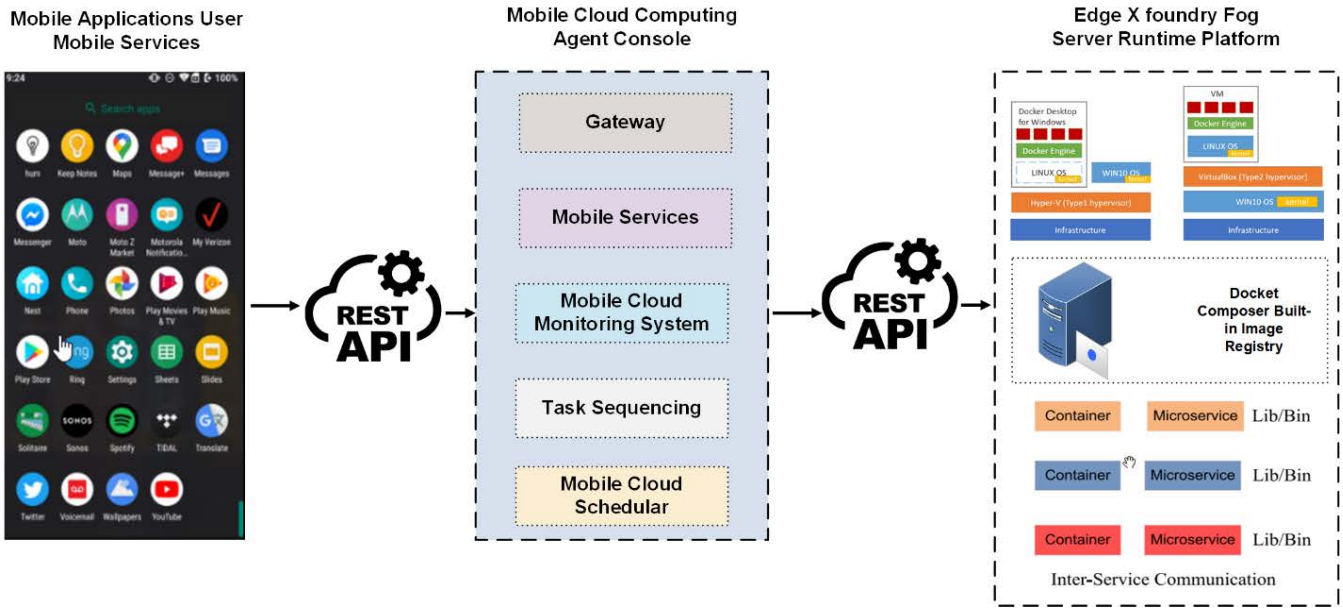


FIGURE 5. MSCMCC implementation uses REST API to offload the task towards mobile cloud computing agent console. These tasks are then forwarded to edge X foundry fog server runtime platform for final task processing on the MCC VM server.

the Edge X Foundry through an open-source platform. The implementation of the MSCMCC framework consisted of three main components. Mobile Users layer, Mobile Cloud Agent Control layer, and Mobile Cloud Resources Layer. The mobile applications offload their related tasks to Mobile Cloud Computing Agent Console through REST API. The JSON format is used to interpret the requests and responses from Mobile Cloud Computing Agent through a Gateway Interface. The console interface reads the API request proximately. Based on the characteristic of the offloaded task, the device services inform the Mobile Cloud Computing Agent what type of services are currently required to execute that particular task. The responsibility of the Mobile Cloud Monitoring System is to check the tasks list and monitor the stability of the system. Task Sequencing sequence the tasks into some logical order, and Task schedular schedule tasks to heterogeneous Mobile Cloud servers for execution. The Run Time is a system operational environment based on the system scenario. Java Runtime Virtual Machine (JVM) runs the Java program effectively. JVM is just like Windows Docker Virtual Machine. Autonomous Microservices are created based on these containers. The containers are registered with a Mobile Cloud server through registry services to consume all the services efficiently. REST API is beneficial to inter-services communication among microservices to achieve lower overhead.

D. COMPARISON OF OFFLOADING FRAMEWORKS

The proposed Microservices Container-Based Mobile Cloud Computing offloading framework provides a lower bootup time than heavyweight VM frameworks. The proposed system effectively improves resource utilization as compared

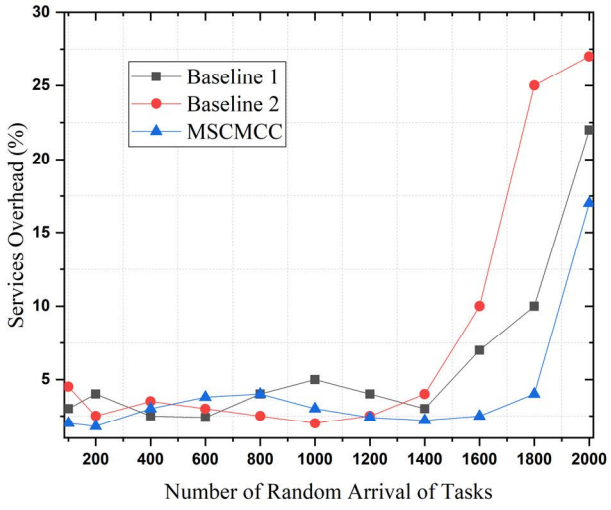


FIGURE 6. Boot time of microservices to compare the random arrival of tasks with the percentage of services overhead over 2000 tasks.

to existing techniques. These aspects are proved through simulation through mentioned parameters from Table 3. Figures 6 and 7 show the boot time results of the microservices over 2000 and 3000 tasks separately. The experimental results show that the MSCMCC technique effectively enhances the overhead of the service for the arrival of the tasks in percentage value.

The proposed approach effectively shows less time for microservices-based applications., In Figures 8 and 9, we design to implement the CPU utilization of resources over 2000 and 300 random arrivals of the tasks. The tasks in MSCMCC show less CPU usage of around 22% compared to existing methodologies.

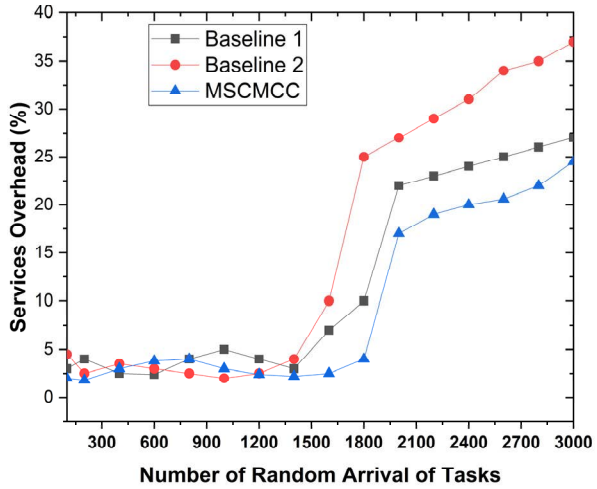


FIGURE 7. Boot time of microservices to compare the random arrival of tasks with the percentage of services overhead over 3000 tasks.

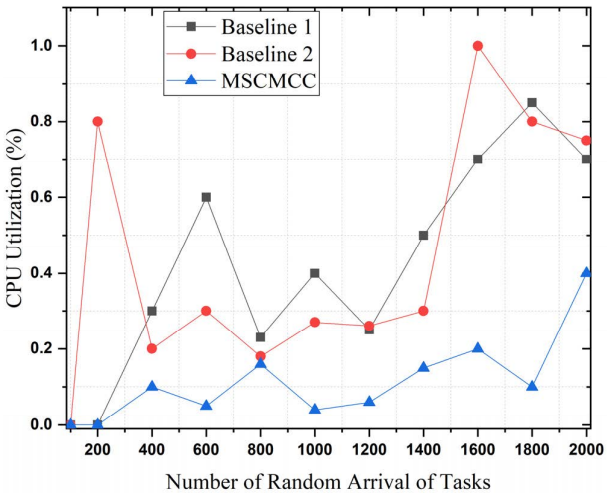


FIGURE 8. CPU utilization of resources for random arrival of tasks for the processing resources for 2000 tasks.

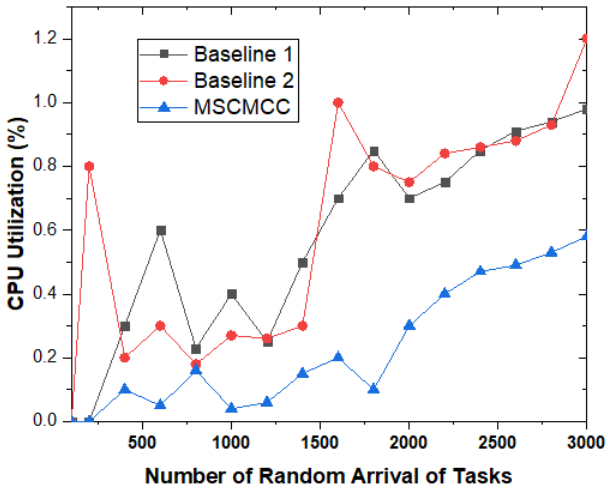


FIGURE 9. CPU utilization of resources for random arrival of tasks for the processing resources for 3000 tasks.

Moreover, figure 10 and 11 shows the improvement in boot time. The MSCMCC shows less boot time than the existing technique by 17% than existing techniques.

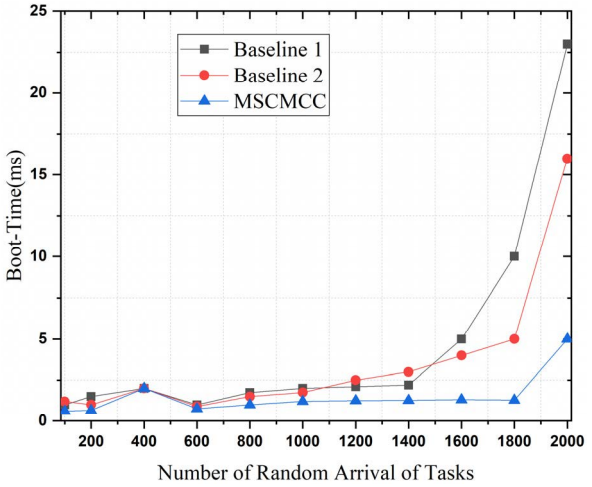


FIGURE 10. Overhead of microservices according to boot time in ms with the number of random arrival of tasks for 2000 tasks.

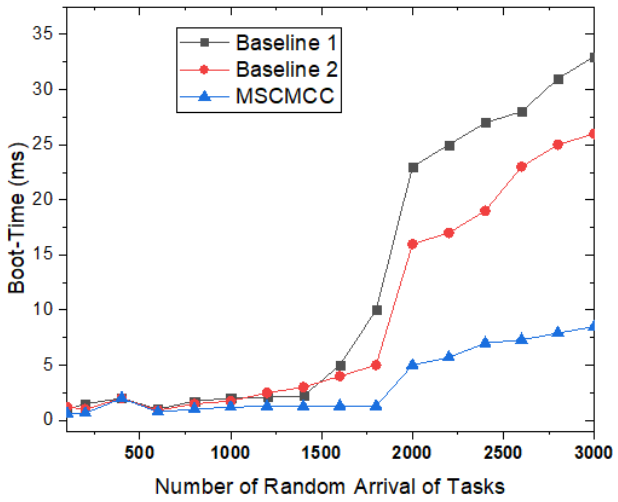
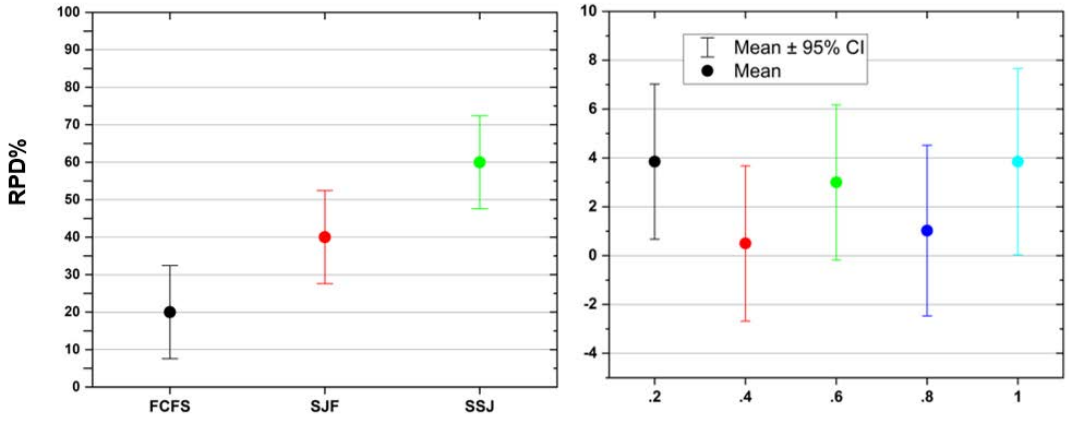


FIGURE 11. Overhead of microservices according to boot time in ms with the number of random arrival of tasks for 3000 tasks.

The results show that the MSCMCC framework effectively improves the overhead, bootup time, and resource utilization. The main reason behind the effective results is the lightweight VM utilization over heavyweight when running Mobile Applications during scheduling. Therefore, our task scheduling framework is efficient for delay-sensitive mobile applications. The proposed system effectively minimizes the cost of healthcare applications by 25%, augmented reality by 23%, E-Transport tasks by 21%, and 3-D games tasks by 19%, the average boot-time of microservices applications by 17%, resource utilization by 36%, and tasks arrival time by 16%.

E. TASK SEQUENCING

Task Sequencing rules such as FCFS, SJF, and SSJ are the main components used to organize the tasks in sequential order for scheduling. Figure 12(a) demonstrates Task Sequencing Rules' working, and figure 12(b) demonstrates the Mean plot of alpha with 95% of HSD Interval and the Mean plot for random Mobile Tasks arrival with



(a) Task Sequencing Rules (b) The mean plot of α with 95% USD Intervals

FIGURE 12. Mean plot of alpha with 95% of HSD interval and mean plot for random mobile tasks arrival with 95% of HSD intervals.

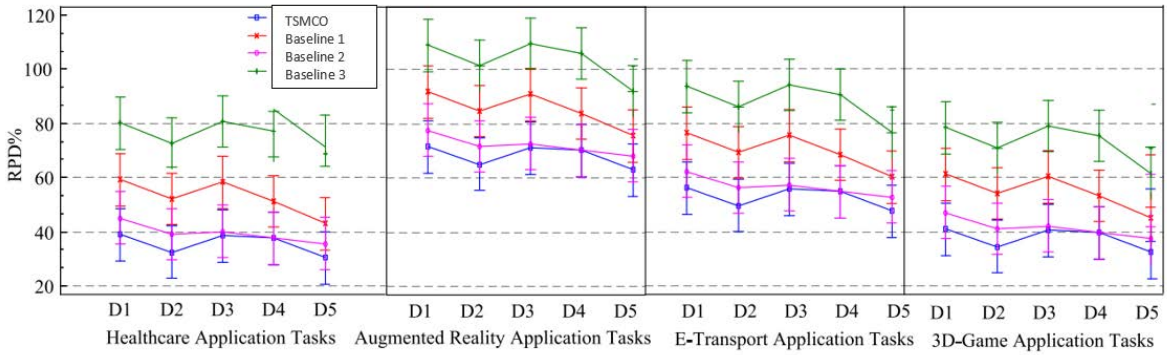


FIGURE 13. Deadlines of healthcare, augmented reality, E-transport, and 3-D game application for RPD%.

95% of HSD intervals. The proposed approach effectively compares the results among FCFS, SJF, and SSJ, which effectively compares and extends results for RPD value in percentage. The proposed technique uses SSJ, which effectively improves the results compared to existing methods for proper implementation of the results. The results show that our proposed approach effectively enhances the proposed technique's working with an effective scenario.

Figure 12(b) demonstrates that RPD's significance for SSJ is less than SSTF and SPF for practical elaboration. Moreover, it accomplished all the scheduled tasks according to the rules of task sequencing defined in the proposed system. The results show a lower delay in the MCC server in a heterogeneous environment. The system dynamically chooses the sequencing of the tasks in the Mobile Clou environment. The selected task sequencing rules systematically choose the sequencing methodology based on the primitive tasks sequencing technique. The task priorities are set for the tasks, which shows the dynamic topological order for the sequencing of the tasks.

F. TASK SCHEDULING

Cost Efficiency in MCC is a key for Mobile Applications. We consider task deadlines and application costs. This research aims to execute the tasks under their deadlines and minimize the mobile application cost. Figures 13 and 14 illustrate the TSMCO framework that incurs lower mobile application execution costs than baseline 1 to 3. The baselines are the existing technologies that elaborate the proposed approach's effectiveness.

Moreover, the results are compared for different applications designed using a microservices-based implementation scheme. The mobile applications run under their deadlines which are shown in figure 13. The proposed system effectively improves the proposed system through iterative features. The solution is continuous until the final optimal solution has been reached.

G. TASK FAILURE RATIO

Our task scheduling mechanism improves the task failure ratio compared to existing techniques. The existing techniques only consider the basic approach for task scheduling. Figures 15 and 16 show the task failure ratio in the proposed technique compared to existing techniques. The TSMCO technique effectively overcame the task failure ratio. This shows that the proposed techniques perform better in the dynamic allocation of Mobile Cloud Servers.

The Proposed TSMCO framework works efficiently in the dynamic allocation of resources and situations. The cost of the system significantly improved and provided the deadlines

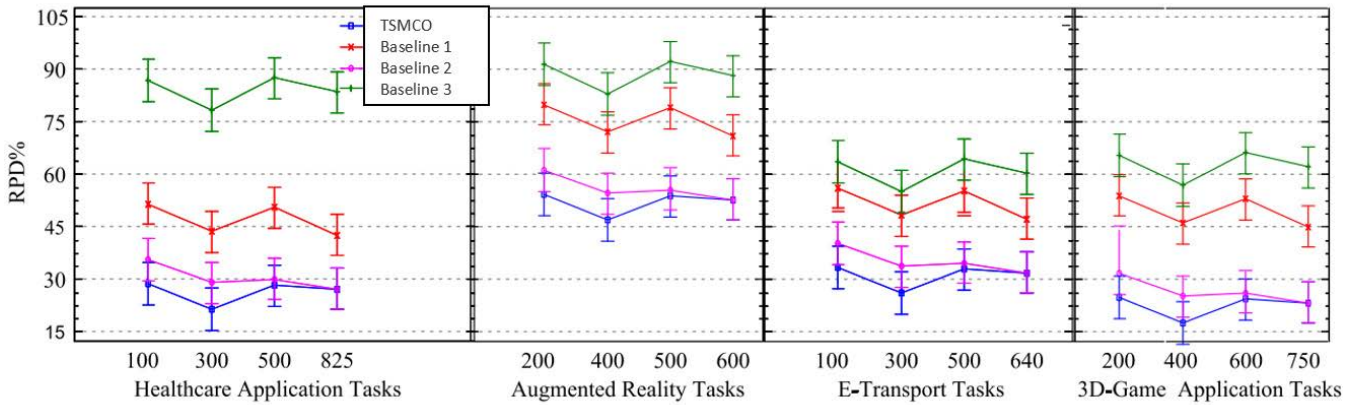


FIGURE 14. The comparison of TSMCO with three other techniques to compare the total cost consumed by applications for healthcare (100 to 825 tasks), augmented reality (200 to 600 tasks), E-transport (100 to 640 tasks), and 3D-games (200 to 750 tasks) for RPD%.

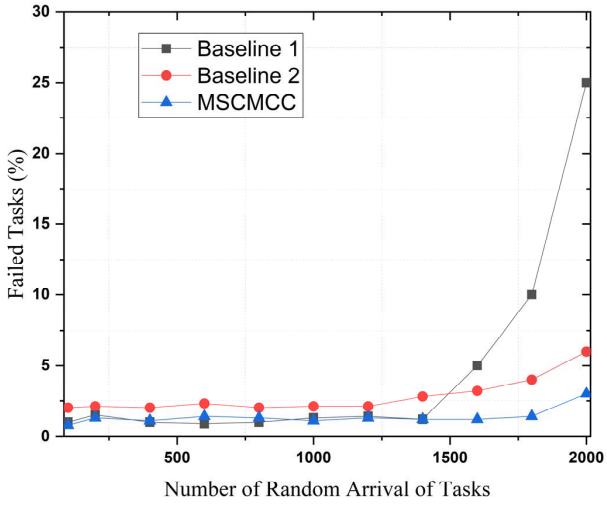


FIGURE 15. Tasks failure ratio during task scheduling with failed ratio.

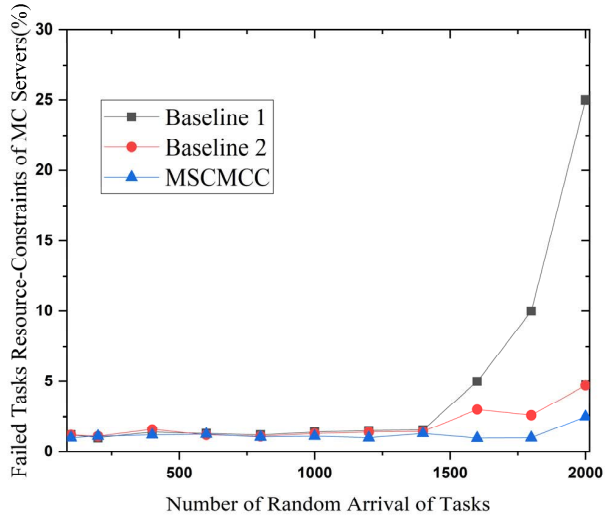


FIGURE 16. Task failure due resource-constrained mobile cloud servers.

for the efficient resources constraints. The Proposed TSMCO framework executes the mobile tasks due to deadline and expands the execution cost.

V. CONCLUSION AND FUTURE WORK

This research proposes a new task scheduling for a microservices-based mobile cloud computing framework. We propose a new microservices-based Mobile Cloud Computing system MSCMCC to run the microservices applications for efficient delay-sensitive applications and a mobility-aware framework to overcome the cost of the application. Moreover, we introduce the TSMCO framework that effectively solves the task scheduling in steps. The steps are task sequences, resource provision and matching, and task scheduling. The experimental results elaborate on the efficient utilization of the resources used in mobile devices and MCC servers. The boot time of the microservices-based applications is lower than existing techniques. The overhead time reduces towards the provided mechanism. We further find from the study that the overhead time of the microservices-based tasks in the proposed technique is lower. The cost of each of the microservices-based applications used is lower than other techniques in baselines one and two. Furthermore, the experimental results elaborate on the server utilization achieved through MSCMCC and TSMCO schemes. To decrease the latency of mobile cloud servers, the mobile server bootup time and microservices latency are effectively utilized, and server cost is effectively minimized.

In the future, Privacy-aware microservices-based task offloading framework for IoT and mobile applications collectively. Our deployment plan is to deploy such mainstream towards Azure, Amazon, and Google. Both transient failure and security are considered during microservices task offloading frameworks. Furthermore, we plan to implement task scheduling decision-making using machine learning or ANN.

REFERENCES

[1] A. Ali, M. M. Iqbal, H. Jamil, H. Akbar, A. Muthanna, M. Ammi, and M. M. Althobaiti, "Multilevel central trust management approach for task scheduling on IoT-based mobile cloud computing," *Sensors*, vol. 22, no. 1, p. 108, 2022.

[2] A. Amini Motlagh, A. Movaghar, and A. M. Rahmani, "Task scheduling mechanisms in cloud computing: A systematic review," *Int. J. Commun. Syst.*, vol. 33, no. 6, p. e4302, Apr. 2020.

[3] N. Parajuli, A. Alsadoon, P. W. C. Prasad, R. S. Ali, and O. H. Alsadoon, "A recent review and a taxonomy for multimedia application in mobile cloud computing based energy efficient transmission," *Multimedia Tools Appl.*, vol. 79, nos. 41–42, pp. 31567–31594, Nov. 2020.

[4] I. A. Elgendy and R. Yadav, "Survey on mobile edge-cloud computing: A taxonomy on computation offloading approaches," in *Security and Privacy Preserving for IoT and 5G Networks*, vol. 95. Cham, Switzerland: Springer, 2022, pp. 117–158.

[5] V. K. Kaliappan, S. Gnanamurthy, C. S. Kumar, R. Thangaraj, and K. Mohanasundaram, "Reduced power consumption by resource scheduling in mobile cloud using optimized neural network," *Mater. Today: Proc.*, vol. 46, pp. 6453–6458, Jan. 2021.

[6] M. T. J. W. P. C. Quasim, "Resource management and task scheduling for IoT using mobile edge computing," *Wireless Pers. Commun.*, vol. 7, pp. 1–18, 2021.

[7] S. Guo, B. Xiao, Y. Yang, and Y. Yang, "Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2016, pp. 1–9.

[8] X. Wang, K. Wang, S. Wu, S. Di, K. Yang, and H. Jin, "Dynamic resource scheduling in cloud radio access network with mobile cloud computing," in *Proc. IEEE/ACM 24th Int. Symp. Quality Service (IWQoS)*, Jun. 2016, pp. 1–6.

[9] M. Chen, S. Guo, K. Liu, X. Liao, and B. Xiao, "Robust computation offloading and resource scheduling in cloudlet-based mobile cloud computing," *IEEE Trans. Mobile Comput.*, vol. 20, no. 5, pp. 2025–2040, May 2021.

[10] X. Liu, Y. Li, and H.-H. Chen, "Wireless resource scheduling based on backoff for multiuser multiservice mobile cloud computing," *IEEE Trans. Veh. Technol.*, vol. 65, no. 11, pp. 9247–9259, Nov. 2016.

[11] Y. Xie, Y. Wang, Y. Jiang, Z. Peng, and Y. Wang, "Multi-objective task scheduling algorithm based on harmony search for grid microservice optimization," in *Proc. J. Phys., Conf.*, vol. 2021, vol. 1746, no. 1, Art. no. 012040.

[12] P. Akki and V. Vijayarajan, "Energy efficient resource scheduling using optimization based neural network in mobile cloud computing," *Wireless Pers. Commun.*, vol. 114, no. 2, pp. 1785–1804, Sep. 2020.

[13] A. Ali, M. M. Iqbal, H. Jamil, H. Akbar, A. Muthanna, M. Ammi, and M. M. Althobaiti, "Multilevel central trust management approach for task scheduling on IoT-based mobile cloud computing," *Sensors*, vol. 22, no. 1, p. 108, Dec. 2021.

[14] A. Ali, M. M. Iqbal, H. Jamil, F. Qayyum, S. Jabbar, O. Cheikhrouhou, M. Baz, and F. Jamil, "An efficient dynamic-decision based task scheduler for task offloading optimization and energy management in mobile cloud computing," *Sensors*, vol. 21, no. 13, p. 4527, Jul. 2021.

[15] A. Lakhan, M. S. Memon, M. Elhoseny, M. A. Mohammed, M. Qabulio, and M. J. C. C. Abdel-Basset, "Cost-efficient mobility offloading and task scheduling for microservices IoVT applications in container-based fog cloud network," in *Cluster Comput.*, vol. 6, pp. 1–23, 2021.

[16] G. Fan, L. Chen, H. Yu, and W. Qi, "Multi-objective optimization of container-based microservice scheduling in edge computing," *Comput. Sci. Inf. Syst.*, vol. 18, no. 1, pp. 23–42, 2020.

[17] L. Abualigah, A. Diabat, S. Mirjalili, M. A. Elaziz, and A. H. Gandomi, "The arithmetic optimization algorithm," *Comput. Methods Appl. Mech. Eng.*, vol. 376, Apr. 2021, Art. no. 113609.

[18] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 945–953.

[19] A. A. Heidari, S. Mirjalili, H. Faris, I. Aljarah, M. Mafarja, and H. Chen, "Harris hawks optimization: Algorithm and applications," *Future Gener. Comput. Syst.*, vol. 97, pp. 849–872, Aug. 2019.

[20] I. Attiya, M. Abd Elaziz, and S. Xiong, "Job scheduling in cloud computing using a modified Harris hawks optimization and simulated annealing algorithm," *Comput. Intell. Neurosci.*, vol. 2020, pp. 1–17, Mar. 2020.

[21] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct./Dec. 2009.

[22] S. Goundar, A. Bhardwaj, and S. Chand, "Task offloading concept using cloud simulations in mobile computing," *Int. J. Syst., Control Commun.*, vol. 12, no. 3, pp. 243–263, 2021.

[23] M. A. Sahito and A. Kehar, "Dynamic content enabled microservice for business applications in distributed cloudlet cloud network," *Int. J.*, vol. 9, no. 7, pp. 1035–1039, 2021.

[24] A. Lakhan, Q.-U.-A. Mastoi, M. Elhoseny, M. S. Memon, and M. A. J. E. I. S. Mohammed, "Deep neural network-based application partitioning and scheduling for hospitals and medical enterprises using IoT assisted mobile fog cloud," *Enterprise Inf. Syst.*, vol. 5, pp. 1–23, 2021.

[25] X. Ma, A. Zhou, S. Zhang, Q. Li, A. X. Liu, and S. Wang, "Dynamic task scheduling in cloud-assisted mobile edge computing," *IEEE Trans. Mobile Comput.*, Sep. 24, 2021, doi: 10.1109/TMC.2021.3115262.

[26] A. M. Rahmani, M. Mohammadi, A. H. Mohammed, S. H. T. Karim, M. K. Majeed, M. Masdari, and M. Hosseinzadeh, "Towards data and computation offloading in mobile cloud computing: Taxonomy, overview, and future directions," *Wireless Pers. Commun.*, vol. 119, pp. 147–185, Feb. 2021.

[27] R. B. Mulinti and M. Nagendra, "An efficient latency aware resource provisioning in cloud assisted mobile edge framework," *Peer–Peer Netw. Appl.*, vol. 14, no. 3, pp. 1044–1057, May 2021.

[28] A. Lakhan, M. A. Dootio, A. H. Sodhro, S. Sandeep, T. M. Groenli, M. S. Khokhar, and L. Wang, "Cost-efficient service selection and execution and blockchain-enabled serverless network for internet of medical things," *Math. Biosci. Eng.*, vol. 18, no. 6, pp. 7344–7362, 2021.

[29] X. Ma, H. Xu, H. Gao, and M. Bian, "Real-time multiple-workflow scheduling in cloud environments," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 4, pp. 4002–4018, Dec. 2021.

[30] A. Lakhan, M. A. Dootio, T. M. Groenli, A. H. Sodhro, and M. S. Khokhar, "Multi-layer latency aware workload assignment of E-Transport IoT applications in mobile sensors cloudlet cloud networks," *Electronics*, vol. 10, no. 14, p. 1719, Jul. 2021.

[31] N. Chaurasia, M. Kumar, R. Chaudhry, and O. P. Verma, "Comprehensive survey on energy-aware server consolidation techniques in cloud computing," *J. Supercomput.*, vol. 77, pp. 11682–11737, Mar. 2021.

[32] L. Abualigah, A. Diabat, P. Sumari, and A. H. Gandomi, "Applications, deployments, and integration of internet of drones (IoD): A review," *IEEE Sensors J.*, vol. 21, no. 22, pp. 25532–25546, Nov. 2021.

[33] A. Samanta and Z. Chang, "Adaptive service offloading for revenue maximization in mobile edge computing with delay-constraint," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 3864–3872, Apr. 2019.

[34] A. Samanta and J. Tang, "Dyme: Dynamic microservice scheduling in edge computing enabled IoT," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 6164–6174, Jul. 2020.

[35] A. Samanta, F. Esposito, and T. G. Nguyen, "Fault-tolerant mechanism for edge-based IoT networks with demand uncertainty," *IEEE Internet Things J.*, vol. 8, no. 23, pp. 16963–16971, Dec. 2021.

[36] J. Lee and J. Gil, "Adaptive fault-tolerant scheduling strategies for mobile cloud computing," *J. Supercomput.*, vol. 75, no. 8, pp. 4472–4488, Aug. 2019.

[37] D. N. Raju and V. Saritha, "Architecture for fault tolerance in mobile cloud computing using disease resistance approach," *Int. J. Commun. Netw. Inf. Secur.*, vol. 8, no. 2, p. 112, 2016.

[38] S. K. Abd, S. A. R. Al-Haddad, F. Hashim, A. B. H. J. Abdullah, and S. Yussof, "Energy-aware fault tolerant task offloading of mobile cloud computing," in *Proc. 5th IEEE Int. Conf. Mobile Cloud Comput., Services, Eng. (MobileCloud)*, Apr. 2017, pp. 161–164.

[39] J. Park, H. Yu, H. Kim, and E. Lee, "Dynamic group-based fault tolerance technique for reliable resource management in mobile cloud computing," *Concurrency Comput., Pract. Exper.*, vol. 28, no. 10, pp. 2756–2769, 2016.

[40] M. M. Al-Sayed, S. Khattab, and F. A. Omara, "Prediction mechanisms for monitoring state of cloud resources using Markov chain model," *J. Parallel Distrib. Comput.*, vol. 96, pp. 163–171, Oct. 2016.

[41] B. Keshanchi, A. Souri, and N. Navimipour, "An improved genetic algorithm for task scheduling in the Cloud environments using the priority queues: Formal verification, simulation, and statistical testing," *J. Syst. Softw.*, vol. 124, pp. 1–21, Feb. 2017.

[42] H. Peng, W.-S. Wen, M.-L. Tseng, and L.-L. Li, "Joint optimization method for task scheduling time and energy consumption in mobile cloud computing environment," *Appl. Soft Comput.*, vol. 80, pp. 534–545, Jul. 2019.
[43] C. Tang, M. Hao, X. Wei, and W. Chen, "Energy-aware task scheduling in mobile cloud computing," *Distrib. Parallel Databases*, vol. 36, no. 3, pp. 529–553, Sep. 2018.
[44] X. Lin, Y. Wang, Q. Xie, and M. Pedram, "Energy and performance-aware task scheduling in a mobile cloud computing environment," in *Proc. IEEE 7th Int. Conf. Cloud Comput.*, Jun. 2014, pp. 192–199.
[45] X. Wei, J. Fan, Z. Lu, and K. Ding, "Application scheduling in mobile cloud computing with load balancing," *J. Appl. Math.*, vol. 2013, Nov. 2013, Art. no. 409539.
[46] A. Lakhan, D. K. Sajnani, M. Tahir, M. Aamir, and R. Lodhi, "Delay sensitive application partitioning and task scheduling in mobile edge cloud prototyping," in *Proc. Int. Conf. 5G Ubiquit. Connectivity*, Cham, Switzerland: Springer, 2018, pp. 59–80.
[47] L. Abualigah, D. Yousri, M. Abd Elaziz, A. A. Ewees, M. A. A. Al-Qaness, and A. H. Gandomi, "Aquila optimizer: A novel meta-heuristic optimization algorithm," *Comput. Ind. Eng.*, vol. 157, Jul. 2021, Art. no. 107250.
[48] X. Ma, S. Zhang, W. Li, P. Zhang, C. Lin, and X. Shen, "Cost-efficient workload scheduling in cloud assisted mobile edge computing," in *Proc. IEEE/ACM 25th Int. Symp. Quality Service (IWQoS)*, Jun. 2017, pp. 1–10.
[49] S. Wu, C. Niu, J. Rao, H. Jin, and X. Dai, "Container-based cloud platform for mobile computation offloading," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2017, pp. 123–132.
[50] A. Lakhan and X. Li, "Transient fault aware application partitioning computational offloading algorithm in microservices based mobile cloudlet networks," *Computing*, vol. 102, no. 1, pp. 105–139, 2020.
[51] M. Goudarzi, H. Wu, M. Palaniswami, and R. Buyya, "An application placement technique for concurrent IoT applications in edge and fog computing environments," *IEEE Trans. Mobile Comput.*, vol. 20, no. 4, pp. 1298–1311, Apr. 2021.
[52] L. Abualigah and A. Diabat, "A novel hybrid antlion optimization algorithm for multi-objective task scheduling problems in cloud computing environments," *Cluster Comput.*, vol. 24, no. 1, pp. 205–223, Mar. 2021.

ABID ALI received the M.S. degree in CS from the University of Engineering and Technology, Taxila, Pakistan, in 2018, where he is currently pursuing the Ph.D. degree in computer science with the Department of Computer Science. He is currently serving as a Lecturer in computer science with the Higher Education Department, KPK, Pakistan. He has eight years of teaching and four years of research experience. He has been a member of multiple committees related to academic fields as well as a participant in research projects. Moreover, he has taught several courses and supervised several capstone projects. His current research interests include the IoT, distributed computing, VANET, security, big data, task scheduling, data mining, cloud and mobile cloud computing, and ICN/SDN. He acted as a reviewer for several journals such as *Fuzzy Sets and Systems* (Elsevier), *Journal of Medical Imaging and Health Informatics*, *CMC* journal, and the IEEE INTERNET OF THINGS JOURNAL.

MUHAMMAD MUNWAR IQBAL received the M.S. degree in computer science from the COMSATS Institute of Information Technology, Lahore, Pakistan, in 2011, the M.Sc. degree in computer science from the University of the Punjab, Lahore, and the Ph.D. degree from the Department of Computer Science Engineering, University of Engineering and Technology, Lahore, under the supervision of Dr. Yasir Saleem. He is currently an Assistant Professor with the Department of Computer Science, University of Engineering and Technology, Taxila, Pakistan. He has authored or coauthored journal and conference papers at the national and international level in the field of computer science. His research interests include machine leaning, databases, semantics web, e-learning, and AI.