Pervasive and Mobile Computing 92 (2023) 101785

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

journal homepage: www.elsevier.com/locate/pmc

An efficient dynamic decision-based task optimization and scheduling approach for microservice-based cost management in mobile cloud computing applications

Mahmood ul Hassan [a], Amin A. Al-Awady [a], Abid Ali [b,*], Muhammad Munawar Iqbal [c], Muhammad Akram [d], Jahangir Khan [e], Ali Ahmad AbuOdeh [a]

[a] Department of Computer Skills, Deanship of Preparatory Year, Najran University, Najran, Kingdom of Saudi Arabia
[b] Department of Computer Science, University of Engineering and Technology, Taxila 47080, Pakistan
[c] Department of Computer Science, Govt. ANK(S) Degree College, KTS 22620, Haripur, Pakistan
[d] Department of Computer Science, College of Computer Science and Information Systems, Najran University, Najran, Kingdom of Saudi Arabia
[e] Department of Computer Science, Applied College Mohyail Asir, King Khalid University, Kingdom of Saudi Arabia

ARTICLE INFO

Article history:
Received 22 November 2021
Received in revised form 23 March 2023
Accepted 1 April 2023
Available online 6 April 2023

Keywords:
Cloud computing
Mobile cloud computing
Task offloading
Task sequencing
Task scheduling
Microservices

ABSTRACT

The use of smartphones and mobile devices has increased significantly, as have Mobile Cloud Applications based on cloud computing. These applications are used in various fields, including Augmented Reality, E-Transportation, 2D/3-D Games, E-Healthcare, and Education. While existing cloud-based frameworks provide such services on Virtual Machines, they incur problems such as overhead, lengthy boot time, and high costs. To address these issues, the paper proposes a Dynamic Decision-Based Task Scheduling Approach for Microservice-based Mobile Cloud Computing Applications (MSCMCC) that can run delay-sensitive applications and mobility with less cost than existing approaches. The study focuses on Task Offloading problems on heterogeneous Mobile Cloud servers. It proposes a Task Offloading and Microservices based Computational Offloading (TSMCO) framework to solve Task Scheduling in steps such as Resource Matching, Task Sequencing, and Task Offloading. Experimental results show that the proposed MSCMCC and TSMCO enhance Mobile Server Utilization while minimizing costs and improving boot time, resource utilization, and task arrival time for various applications. Specifically, the proposed system effectively reduces 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%.

1. Introduction

Edge Computing and Mobile Cloud Computing (MCC) enhance the processing capabilities of mobile devices. Due to the limitations of mobile devices, such as battery life, processing power, resource utilization, and cost, task scheduling

* Corresponding author.
E-mail addresses: mahmood.mscs@gmail.com (M.u. Hassan), aaalawady@nu.edu.sa (A.A. Al-Awady), abidali.hzr@gmail.com (A. Ali), munwariq@gmail.com (M.M. Iqbal), akram.moghal@gmail.com (M. Akram), jhkhan@kku.edu.sa (J. Khan), ali.abuodeh@windowslive.com (A.A. AbuOdeh).

https://doi.org/10.1016/j.pmcj.2023.101785
1574-1192/

is a crucial aspect of mobile computing. Applications like 2D/3D E-Gaming, Augmented Reality, E-Healthcare, and E-Transport require significant modifications to be processed efficiently on mobile devices [1,2]. Contemporary mobile devices can gather, handle, and dispatch data even in unfavorable conditions for data collection. The data processing environment relies on shared characteristics. The motivation for implementing MCC is to broaden the potential of mobile devices and incorporate task scheduling capabilities within them [3]. Due to their demanding operational requirements, mobile devices encounter challenges such as limited CPU, low bandwidth, and battery power. As a solution, these devices delegate their tasks to Mobile Cloud Computing (MCC) to alleviate the burden on their computational and power resources. MCC is interconnected with Edge Computing, Fog Computing, and Cloudlet. This study focused on the task offloading of microservices-based applications in MCC. In [4], a proposed framework and model aim to improve processing performance and enhance application capabilities to save battery life for devices with limited reserves. This framework focuses on offloading tasks to achieve these goals [5,6]. Using MCC to regulate execution performance, the suggested framework amplifies application effectiveness and advances work quality. In a study conducted in [7], the research findings show that the boot time of microservices-based Virtual Machines (VMs) is 28 s, which is a significant delay and adds to the processing overhead of these applications. The limitations highlighted in this study provide insights into the feasibility of using a microservices-based task offloading framework, as explored in this research [8].

1.1. Motivation

Over the past few years, resource-intensive and battery-consuming applications have become increasingly prevalent in mobile devices, including online games, cloud-based apps, and other demanding applications. These applications require minimal local services to be executed correctly and quickly. At the same time, user-centric applications require the support of heavy virtual machines (VMs), which can be costly due to their usage-based payment model. Mobility, cost, and interactivity are the main challenges for the existing mobile cloud computing (MCC) paradigm in providing these services. Additionally, scheduling resources efficiently for mobile tasks is a significant challenge for MCC in microservices-based applications [9]. Scheduling tasks is a crucial aspect of mobile cloud computing as it poses significant challenges related to the limited capabilities of mobile devices, storage constraints, processing capabilities, and network bandwidth requirements. Conversely, cloud-based mobile cloud computing has vast processing abilities, unrestricted bandwidth, and no storage restrictions, making it necessary to employ task scheduling for mobile-based microservices applications.

This paper discusses the problem of cost-efficient task scheduling in the context of Mobile Cloud Computing (MCC) for Internet of Things (IoT) applications. Specifically, the focus is on the MCC-based cloud network, intending to reduce the cost of mobile application services. The study concentrates on the computational and communication costs associated with task scheduling and offloading, common issues in mobile applications that comprise independent and fine-grained sub-tasks. Fine-grained refers to the independent nature of each task, with specific attributes and data that make the workload efficient. Each task is associated with vector attributes such as CPU instructions, data size, and execution deadline, while the MCC services are evaluated based on their price and speed. The research considers several questions during the task scheduling process.

(i) How to select the cost-efficient MCC VM for every task to meet the processing requirements of every task?
(ii) How are offloaded tasks executed sequentially before scheduling on MCC?
(iii) How to accomplish optimal task scheduling to reduce and execute the tasks under the operating costs and defined deadlines?

1.2. Summary/contributions

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

(i) Our proposed method, called Microservices Container-based MCC system (MSCMCC), facilitates computational offloading between mobile devices and resource-rich MCC models. To improve the performance, we implemented a Docker container that reduces the overhead for services and lowers the VMs' boot time.
(ii) While considering MCC servers with different attributes, we also considered the individual quality-of-service requirements for each task. We selected the MCC servers with VMs that met the service demand and developed a service-matching algorithm to execute instructions based on service requirements and tasks.
(iii) We adopted the FCFC and SJF task sequence algorithm to schedule randomly generated tasks from mobile devices and put slack time and task size in sequences.
(iv) The computational cost and communication time for scheduling fine-grained tasks from local mobile devices towards MCC VMs pose a significant challenge. Therefore, we proposed a microservices-based cost-efficient task scheduler algorithm that reduces the overhead cost of mobile tasks to MCC servers.
(v) Our novel contribution involves sequencing tasks based on their priority, with tasks requiring special consideration during offloading given the highest priority.

The remaining sections of this paper are structured as follows: In Section 2, previous research on task scheduling and fault tolerance is reviewed. Section 3 outlines our research approach and offers solutions to the problems discussed in Section 1. Section 4 presents our proposed method and its simulations using an MCSMCC approach. Finally, Section 5 concludes our proposed system, emphasizing its simplicity in fault tolerance methodology.

2. Literature review

The popularity of computational offloading of mobile applications among mobile cloud users has increased with the rise of mobile devices and mobile applications. This method enables mobile devices with computationally intensive tasks to shift their workload to the cloud for execution on mobile cloud servers. [10]. Outsourcing mobile tasks is one way to enhance the performance of mobile devices and their batteries. To achieve this, mobile cloud support transfers heavy-duty application tasks away from the mobile device. Previous research has focused on optimizing mobile application performance while reducing computational costs and conserving the processing power of mobile devices [11].

In [12], the authors' CloneCloud framework aims to enhance the battery life and execution speed of mobile devices by leveraging Centric Cloud Servers to handle computationally intensive tasks. In the paper [13], the proposed framework by the authors aims to conserve the mobile device's battery by only running computationally intensive tasks on it. Another approach, MAUI, uses profiling technologies to determine whether to execute the task locally or on the mobile cloud server. The key distinction is to make the optimal decision for offloading the task based on runtime. The primary goal of this approach is to efficiently offload the task during its runtime.

As far as we know, there has not been a microservices-based MCC framework introduced yet that caters to time-sensitive and intricate tasks. In this context, we suggest a new MCSMCC mobile cloud computing framework designed to perform tasks with the least energy and cost. Furthermore, we present a cost-conscious framework CCCOF that aims to optimize the cost-centric and computational offloading procedures. CCCOF ensures the quality of service by running services within the given time constraints while keeping the costs of resources at a minimum. In [14], the ThinkAir approach, introduced by the authors, offers a way to transfer mobile device tasks to the mobile cloud through a computational offloading framework. This can be achieved by utilizing mobile virtualization technology and facilitating task offloading. The authors also developed meta-heuristic algorithmic methods to tackle task-scheduling challenges in conjunction with the existing task-scheduling frameworks [15,16].

The resolution offered by computation offloading frameworks is unsustainable due to cloud network latencies. As a result, there is a greater need for problem-oriented cloudlet-based computational offloading resources that cater to the latency challenges of wireless-based access networks. Cloudlet frameworks have addressed this issue by bringing mobile devices closer to computational resources. In [17], the virtual machine-based Cloudlet framework proposed by Satyanarayana et al. offers elasticity, scalability, and mobility. This framework brings Cloudlets closer to mobile phones, serving as a single hop toward cell phones. Another researcher presented Rattrap [18], a mobile cloud-centric environment that was proposed to offload the computational tasks of mobile devices to the cloud using an Android-based system. Virtual machines with containers were utilized in this research to reduce the boot time of cloud platforms' monolithic services. However, while this approach effectively reduces service boot time, it fails to meet the fine-grained resource requirements of mobile devices for resource-intensive applications.

Certain research contributions improve the cost-effectiveness of services provided by mobile applications. In [19], the authors leverage various mobile cloudlet services with varying characteristics to enhance cost-effectiveness. The authors evaluate computation time, communication time, and deadline cost as essential factors. In [20–22], The study investigates cost-efficient and energy-efficient task offloading in mobile cloud computing. Its primary aim is to preserve the battery life of mobile devices by transferring tasks to the mobile cloud. The authors perceive resources as storage and propose a mobile cloud-based computational offloading framework for task scheduling that is presented as an effective computing model. The [23–25] research objective is to analyze the resource consumption pattern of mobile applications in mobile cloud computing. Various cost pricing models, such as spot instances, on-demand, and on-reserved systems, are discussed to ensure efficient collaboration for resource consumption. The primary goal of these studies is to minimize resource rental expenses and deliver mobile services within their designated time frames.

Furthermore, the authors in [26–28] aim to improve the performance of mobile cloud applications by exploring cost-efficient and cost-effective real-time analysis techniques to mitigate the running costs associated with task scheduling. The studies aim to optimize task execution by identifying effective workflow categories and reducing data transfer and execution time. Additionally, the studies consider mobile device services' computational and communication costs during task scheduling in MCC [29]. In [30], the authors employ mobile edge computing to reduce service latency, optimize revenue, and enhance the quality of services. They present the benefits of increased revenue and service utilization. Although the authors enhance the usage, service latency, revenue, and utility value, they do not consider cost management, resource matching, task deadline management, server utilization ratio, and the time to boot up microservices. Authors in [31] aim to create a mathematical model for a microservice scheduling framework that enhances satisfaction, reduces network delay, energy consumption, average price, and failure rate, and improves network throughput. However, the authors do not focus on addressing issues such as service delivery latency, cost management of microservices, resource matching, task failure ratio, service utilization, and task sequencing, as these parameters are outside the scope of their research. In [32], the author introduced an auction mechanism proposed as a technique to model the interaction among Mobile Edge Computing systems. The approach employs a justified methodology to manage offloading tasks and measure performance. Table 1 summarizes the literature, which includes Task Sequencing, Server Utilization, Task Failure Ratio, Handling Task Deadlines, Boot Time, Cost Management, Resource Matching, and Service Delivery. This approach considers these parameters, which were not considered by previous approaches.

This related work section proposes several algorithms, and our proposed methodology aims to tackle the problem of migrating microservices for content-sensitive applications. While multiple techniques are suggested to address this issue,

Table 1
Based on their characteristics, the proposed system can be compared to related work.

Papers	Task sequence	VM server use	Fault tolerance	Deadlines	Load time	Process cost organization	Resource complementary	Latency
Lee et al. (2019)	✔	✔	–	✔	✔	✔	–	–
Raju et al. (2016)	✔	✔	–	–	✔	–	–	✔
Abd et al. (2019)	✔	✔	✔	–	✔	–	–	✔
Park et al. (2016)	✔	✔	✔	✔	–	✔	✔	✔
Al-Syed et al. (2016)	✔	✔	–	–	–	–	–	–
Keshanchi et al. (2017)	✔	–	–	✔	–	–	✔	–
Peng et al. (2019)	✔	–	✔	–	✔	–	✔	–
Tang et al. (2018)	✔	–	✔	–	–	–	–	✔
Lin et al. (2014)	–	–	✔	✔	–	–	–	✔
Guo et al. (2022)	–	–	✔	✔	–	✔	–	–
Wei et al. (2023)	–	✔	–	–	✔	–	–	–
Proposed methodology	✔	✔	✔	✔	✔	✔	✔	✔

they do not consider the computational cost requirements and demands for future needs. In the next section, we will discuss the standard methods used to address the problem and provide a detailed description of the problem.

As the use of microservices increases on mobile devices, the tasks required become more resource-intensive in today's environment. Instead of performing entire tasks, only the necessary operations must be performed. The primary motivation behind offloading microservices-based tasks is to reduce the resources used and increase cost-effectiveness through virtual machine-based resources. Previous studies have focused solely on task scheduling on collective tasks with individual services. However, it is essential to consider cost-effectiveness and resource management for each microservice. This approach offloads tasks to the Mobile Cloud Computing (MCC) virtual machine.

To the best of our knowledge, there has not been any suggestion of a framework for a Dynamic Decision-Based Task Scheduling Approach in Microservice-based Mobile Cloud Computing Applications. This research proposes the MSCMCC framework, which enables microservices-based task execution on mobile devices at a very low cost. The framework ensures Quality of Service for microservices-based mobile device applications over mobile cloud computing. MSCMCC enhances application efficiency by providing an effective resource constraints framework that enables task execution under a deadline while minimizing application cost.

2.1. Problem description

The literature has explored task offloading in mobile cloud computing [25,33]; however, task scheduling has become a concern. To address this issue, we developed the Cost-Centric and Computational Offloading Framework (CCCOF), which considers communication, offloading cost, and processing power during scheduling and offloading. Our paper aims to reduce computation and communication costs while maximizing processing power on both MCC and mobile devices. To address this problem, we propose the MSCMCC framework, which consists of three layers, as shown in Fig. 1. The Task Scheduling Layer, which includes the System Monitoring Agent, Cloud Computing Agent, and Task Sequencing using sequence algorithms, is responsible for scheduling decisions. The MSCMCC framework is higher-order, while TSMCO and CCCOF are lower-order frameworks within the proposed methodology. CCCOF employs a cost model to compute the resource access and cost for microservices-based tasks. The cost is computed for tasks that must be scheduled on the MCC server.

Furthermore, TSMCO has been developed to match resources between offloaded tasks and MCC servers. The advantage of optimizing tasks is that it allows for the pair-wise processing of diverse applications. In the subsequent section of the paper, we elaborate on the proposed architecture for offloading tasks using MSCMCC, CCCOF, and TSMCO.

3. Proposed microservices-based MCC architecture

The MSCMCC framework proposed in Fig. 1 comprises three layers: Mobile Users Layer, Task Scheduling Layer, and Mobile Cloud Layer. Typically, heterogeneous mobile users generate tasks to offload, which are then passed to the Task Scheduling Layer via an API. The Task Scheduling Layer consists of four main modules that receive the offloaded tasks. The Cloud Computing Agent (CCA) manages and handles all offloaded tasks and is supported by the System Monitoring Agent, Task Sequences, and Task Scheduling Handler. Acting as an intermediary between mobile devices and system resources, CCA collects data such as configuration information, metrics, and logs from mobile device APIs. All components of the MCC virtual network, including the System Monitoring Agent, Task Sequences, and Task Scheduling Agent, exist on MCC servers. CCA utilizes the functionality of all three modules to measure performance and manage workload requests.

We have primarily utilized FCFC and SJF algorithms for task sequencing from mobile devices. It is important to note that each task in this scheduling includes vector attributes such as execution time, execution deadline, data size, and CPU instructions. Rules for task sequencing have been proposed based on sorting algorithms to ensure minimum cost overhead rules are utilized. The System Monitoring Agent maintains a table that includes tasks and their resource list,

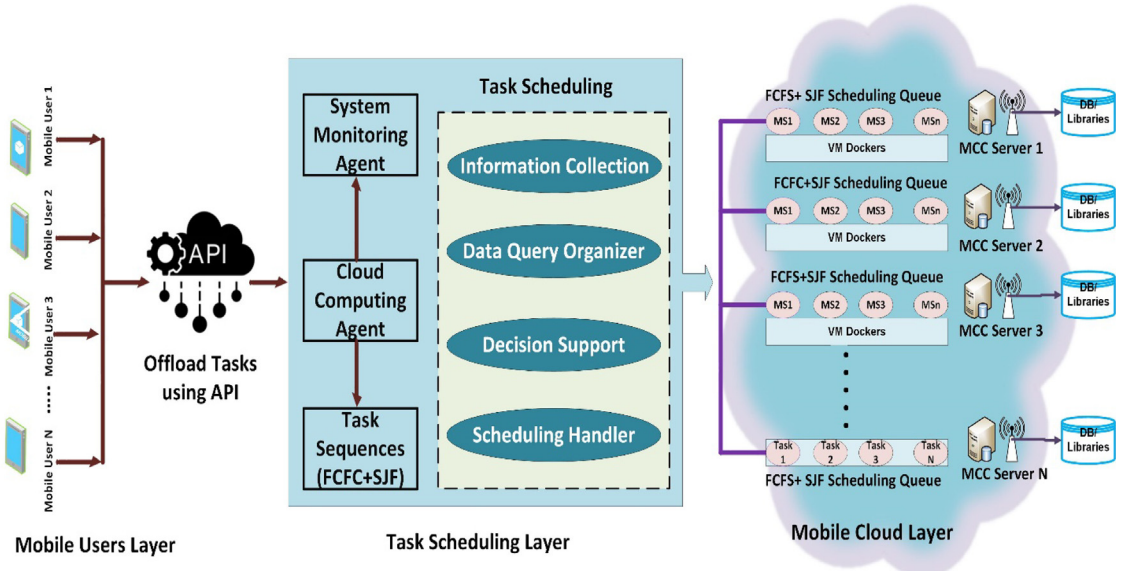

Fig. 1. Proposed MSCMCC system architecture.

updated after every event in the Task Scheduling Layer, with information about resources consumed or left from the system or task completion. In the Mobile Cloud Layer, tasks are swapped among MCC servers, resulting in improved early task scheduling and an enhanced cost function. The task-swapping mechanism among MCC servers enhances resource handling and other task provision mechanisms.

The Mobile Cloud Layer comprises various types of Mobile Cloud servers that utilize VMs to execute and manage resource allocation. These VMs are in the topmost layer of MCC server layers. Within each VM, the MCC Server engine adds or removes microservices as needed to handle tasks. These microservices are managed through containers and communicated via REST-API services by the container MCC Server engine. To load any necessary data or functions required for microservices offloading, DB/Libraries are connected to the MCC servers.

3.1. Task clarification using microservices

The Mobile Devices produced independent tasks from different mobile device applications. These tasks contain data embedded in their applications. 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.

3.2. Tasks mobility

We have integrated the MCC cloud servers at the end of the MCC network. We have introduced a Mobility Module in CCA that allows the MCC network to choose the mobile users or subscribers for packet delivery. One of the packet delivery schemes is Location Management. CCA also manages all mobile subscribers' connections, including the attachment points for the final task scheduling. Handoff Management is a crucial factor that determines the scenario for handoff management. The Task Mobility Section deals with the issues and features of location and handoff management problems. In MCC, Mobile Devices change their location using Mobility options, which trigger their connection from one access point to another. This process is known as Handoff Management. We divide the handoff management into three primary stages in MCC Mobile Device connections. First, the activation process of MCC Mobile Devices is caused by the changing condition of Mobile Devices, the MCC network agent, and the MCC network. Second, when a new Mobile Device connection is established, CCA performs additional routing and discovers a new handoff connection tool. Third, the data is delivered from the old connection points to other connections to support the quality of service operations and handling procedures. The old connection refers to the connection of Mobile Devices in the previous handoff process, while the new connection refers to the handoff towards the new MCC access point.

3.3. Microservices-based MCC servers characterization of resources

The MCC paradigm employs Mobile Cloud Layer containers as a standard methodology for running Mobile Applications in the MCC network. This emerging approach has resulted in significant advancements in managing group containers. The primary challenge, however, is deploying microservices containers through the MCC server layer. Microservices are small, self-contained services that interact with external resources through well-defined APIs. Self-established teams in MSCMCC own these microservices. The approach utilizes heterogeneous mobile cloud servers with VMs on each server to optimize performance. The MCC server engine is responsible for adding or removing microservices quickly. The Mobile Cloud Layer comprises heterogeneous MCC servers with devices and processors with various computational capabilities, runtime, bandwidth, costs, and VMs (container microservices).

3.4. Runtime microservices in MSCMCC

Microservices is a methodology for developing runtime software that involves arranging an application as a series of loosely coupled services, a variation of the service-oriented architecture model. Within microservices, protocols are lightweight, and services are fine-grained. In contrast, MSCMCC utilizes a single VM per MCC server, while a single VM can support multiple microservices at once in the microservices architecture. Every microservice in the microservices architecture runs a single task at a time. CCA employs cost-efficient microservices to handle computational tasks, resulting in efficient outcomes. Each microservice in CCA contains libraries and resources related to its specific task. During task offloading in MSCMCC, mobile application tasks are handled through fine-grained microservices architecture.

3.5. Problem formulation

We began by selecting a set of delay-sensitive tasks, $T = \{t_1, t_2, \ldots\ldots, t_n\}$, to be offloaded to the MCC server. These tasks were then scheduled using the MCC server. Since we were dealing with microservices, each task was accompanied by certain attributes, denoted as $t_i = \{T_w, T_{data}, T_d, T_{storage}\}$. The T_w attribute indicated the task's workload, T_{data} showed the data associated with the task during its transmission from the mobile device to the MCC server VM, T_d represented the task's deadline, and $T_{storage}$ indicated the storage requirements for the task.

Assuming the MCC cloud configuration, we have a set of N MCC servers, denoted as $M = \{m_1, m_2, \ldots, N\}$. Each MCC server m_j has a set of attributes, including B^{wj}_{MCC}, which represents the bandwidth between the MCC Centric Agent and the MCC server for task offloading, ξ_j, which represents the computing rates of the jth MCC server, S_{cj}, which represents the total storage capacity of the MCC server j, and VM^j_{mic}, which represents the deployed VMs for microservices through the MCC server j, capable of handling tasks. Each VM^j_{mic} comprises multiple containers for executing the microservices to perform various tasks, with each microservice having its database and libraries required for execution.

To schedule a task at MCC server j, the required bandwidth t_i is denoted by B^j_i, and the resources, such as Storage, RAM, Bandwidth, etc., are denoted by C_j. The offloaded task cost to MCC server j is denoted by t_i, and the task's state is denoted by $y_j = 1$. Notations for limited page space are shown in this paper, and the remaining notations are illustrated in Table 2. Eq. (1) shows that the binary variable s_{ij} determines whether task t_i is scheduled through MCC server M_j, with s_{ij} being either zero or one for all application tasks. The proposed servers decide on task scheduling.

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

In this problem context, each task is assigned to a specific M_j, and the MCC server schedules tasks concurrently. The task assignment t_i to MCC server M is illustrated in Eq. (2), where the number 1 represents the ready tasks assigned to the MCC server for processing on its VMs. It indicates a positive task assignment to the appropriate server VM out of N tasks.

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

Each MCC server has a finite resource capacity, ensuring that the demands for task offloading do not surpass the server's capabilities. To verify the task scheduling requirements for offloading, Eq. (3) presents the scenario for assessing the task offloading capabilities of s_{cj} and s_{ij} against s_{cj}. The aim is to ensure that no task processing requirements go beyond the limitations of the MCC server.

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

The MCC resource server possesses limited resources, including many virtual machines (VMs) available to execute microservices for coordinating tasks. Therefore, Eq. (4) indicates a decision-making process that assigns fewer computational

Table 2
Notations with description.

Symbols	Description
N	N represents the task set that includes all mobile applications.
M	M refers to the total number of MCC servers in the model.
s_{ij}	The notation s_{ij} indicates that the ith task is assigned to the jth MCC server.
y_j	The notation y_j denotes the state of the MCC server M_j, which can be ON or OFF.
t_i	The notation t_i represents the ith task of the mobile application.
M_j	The notation M_j represents the jth MCC server in the model.
S_{cj}	S_{cj} refers to the storage demand of the tasks t_i.
T_d	T_d represents the deadline for the task t_i.
T_w	T_w represents the computational workload or data of the task t_i.
T_{data}	T_{data} represents the size of the task t_i during transmission from the mobile device to the MCC server.
B_i^j	The notation B_i^j Represents the total bandwidth the task demands to the MCC server M_j.
S_{cj}	The notation S_{cj} represents the capacity of the jth MCC server.
VM_{mic}^j	The notation VM_{mic}^j represents the total number of VMs deployed at the MCC server M_j.
ξ_j	The notation ξ_j represents the computational speed of an individual VM under the MCC server M_j.
B_{MCC}^{wj}	The notation B_{MCC}^{wj} represents the bandwidth between the MCC Centric Agent and the MCC server M_j during the task offloading t_i.
C_j	The notation C_j represents the cost of the jth fog server in the method.

tasks. Each requested task, which consists of microservices, must utilize less computation than the capacity of the VMs.

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

Each designated task, denoted as t_i, is assigned to an optimal MCC server, M_j, by the Cloud Computing Agent. The execution of the task on the assigned server is determined using Eq. (5), which considers the task's resource, storage, and time requirements to select the appropriate server, M_j, for the final decision.

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

As a result of the Cloud Computing Agent's decision to assign a task t_i for computation on the MCC server, the task now has additional options for computational offloading. It can transmit the results to the Mobile Device from the MCC server. The computational process is depicted in Eq. (6).

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

Eq. (6) defines the input and output data sizes of task t_i in terms of $T - Date_i^{enter}$ and $T - Date_i^{leave}$, respectively, after being processed by the MCC server M_j. The bandwidth for uplink and downlink, denoted by $B_{MCC}^{wj_{ij} \ up}$ and $B_{MCC_{ij}}^{wj \ down}$ Respectively, it represents the link rates for offloading data from the mobile device to the MCC server and receiving the results back. The Round-Trip Time (R_T^T) is the time between sending and receiving data for all tasks (microservices). We have measured the bandwidth required for each task, and Eq. (7) shows the bandwidth requirements for each task for R_T^T.

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

Eq. (7) outlines the bandwidth requirements. However, since each task varies, so does the necessary bandwidth. Eq. (8) demonstrates the bandwidth inequality of task t_i. The bandwidth is evaluated based on each VM Time, Data, and computational speed under MCC server M_j. Eq. (8) is utilized to determine the bandwidth inequality.

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

To attain the desired performance of the suggested system, we ensure that each task denoted as t_i, which originates from mobile devices, is completed before its deadline. This approach reduces the overall cost of the MCC server, M_j. The

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

calculation of the necessary communication bandwidth between the MCC server and mobile application is expressed in Eq. (9). Following this, the communication bandwidth for task t_i is determined by evaluating the inequality bandwidth, which determines the actual and anticipated bandwidth required for offloading the task.

$$B^{wij}_{MCC} = \frac{T - \text{data}_i}{T_d - \frac{T^i_w}{\xi_j}} \tag{9}$$

Each MCC server, denoted as M_j, has a finite amount of bandwidth and a limited number of VMs, in addition to the individual task bandwidth. Therefore, the task allocation to M_j must use less bandwidth than the server can handle. Eq. (10) states that the total bandwidth consumed by the tasks should not exceed the bandwidth required by M_j to execute w_j tasks. The variable w_{ij} represents the necessary bandwidth for executing the tasks in the correct order.

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

The Cloud Computing Agent (CCA) functions as a connector and monitor to the MCC server, overseeing its performance regularly. The cost of the MCC server is determined by two primary factors: its state and the resources required for microservices for each offloaded task. CCA handles the tasks of the user's mobile application, and its cost is based on its operational state and the computational capabilities requested from the MCC server. We use the binary variable ∂_j to indicate the status of the MCC Server, as expressed in Eq. (11). This variable reflects whether the server is on or off for task processing.

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

3.6. Cost-Centric and Computational Offloading Framework (CCCOF)

This section outlines the MCC Cost Model, which comprises the Cost-Centric and Computational Offloading Framework (CCCOF) and defines the quality of service. Microservices are not standalone resource computational applications. CCCOF includes a cost model that explains the on-demand resource access method, ensuring connectivity based on the microservices of the business applications framework. Eq. (12) demonstrates the on-demand cost model for mobile applications, which calculates the processing demand for each selected application used in the simulation.

$$C_j = \delta_j \ {}^*s_{ij} \ {}^*T^e_{t_i} \tag{12}$$

Table 3 presents the unit price for computational work, δ_j, for each MCC server. The resource constraints for each server are then computed using Eqs. (13)–(25), which collectively determine the constrained resources for MCC servers MMC_1 to MMC_3. The state of each server is monitored through CPU costs, bandwidth requirements, and microservices-based processing tasks, which must be distributed against every resource. The decision to offload tasks is made using the parameters in Table 3, represented by $S_{ij} = \{0, 1\}$. Eq. (13) calculates each task's minimum required resources (Rc) based on On-Demand δ_j and the MCC Server State ∂_j. Eqs. (14), (16), (17), (18), and (19) are then used to compute the resources demand, the time for resource computation, the task of mobile applications, and the comparison and execution of Cloud-based resources tasks. Eqs. (20) and (21) determine the capacity of the j_{th} MCC server for resources and set it to 1 to show that resources are ready to offload the task to MCC servers. Finally, Eqs. (22), (23), and (24) consider server, bandwidth, and VM-based resources required for each microservices-based task to be offloaded to the cloud. Ultimately, $S_{ij} = \{0, 1\}$ decides to offload the task and the required resources.

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

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

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

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

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

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

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

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

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

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

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

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

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

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. $\sum_{i=1}^{M} \sum_{j=1}^{N} S_{ii} * \partial_j * C_j$ Call(Decision Scheme)
2. $task\_schedule(G, T, M)$
3. $task\_sequence(M, T, G)$
4. $T_{LIST}^{Mobile}[\,] \leftarrow NULL$
5. while(M) do
6. while(G) do
7. while(T)
8. if($T \neq \varphi$) then
9. $resource\_algo(T_{exe}^{time}, T_{exe}^{limit}, T_{data}^{req}, T_{cpu}^{ins})$
10. if($T_{LIST}^{Mobile}[\,] \neq \varphi$) then
11. $task\_sequence(M, T, G)$
12. $task\_sequence(M, T, G)$
13. $R_c * \leftarrow (\sum_{i=1}^{M} \sum_{j=1}^{N} \partial_j * C_j) * \leftarrow T_{LIST}^{Mobile}[\,]$
14. $R_c \leftarrow R_c *$
15. Return R_c;
16. End Loop;

3.7. Task Scheduling and Microservices based Computational Offloading (TSMCO) framework

The TSMCO framework comprises various components and presents a new approach to task scheduling and microservices-based computational offloading. It includes the Resource Matching Algorithm and Task Scheduling Algorithm, which work together to determine the processing of tasks and create a sequential list of completed tasks from

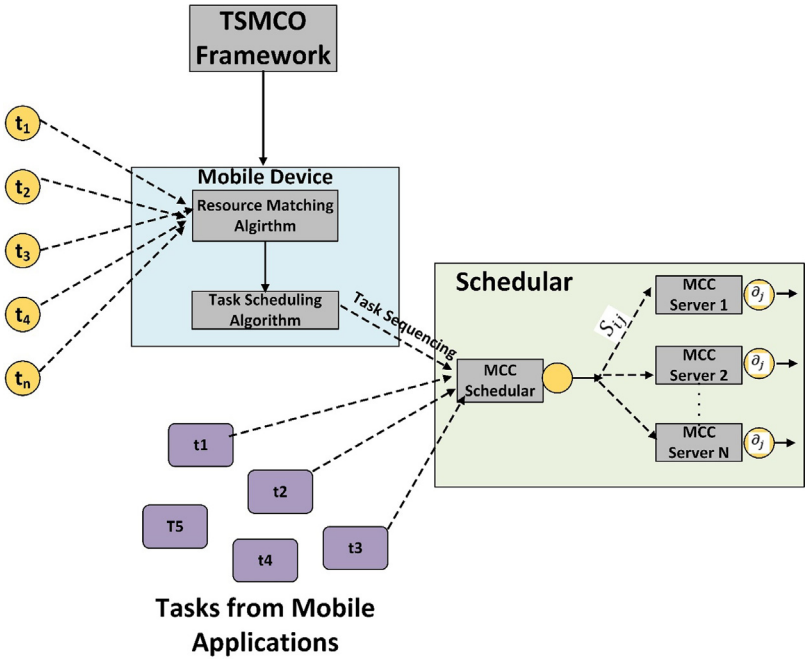

Fig. 2. The TSMCO Framework has been implemented with a Resource Matching Algorithm and Task Scheduling Algorithm to receive microservices-based tasks from mobile devices. This framework is used for task offloading, and the MCC Scheduler utilizes virtual machines to process the offloaded tasks.

mobile devices. The framework offers a resource matching system that matches the required resources of incoming tasks with MCC VM servers, and once a match is found, it proceeds to sequence the tasks. Fig. 2 provides a visual representation of the TSMCO framework components, with the first module being resource matching for incoming mobile device tasks and matching them with MCC servers for processing. Task Sequencing is a crucial component of TSMCO, as it uses sorting algorithms to sort tasks into different categories for cost-effective scheduling. Task sequencing provides input for task scheduling, with the task sequence t_i scheduled on the MCC server MC_j if $S_{ij} = 1$ or $S_{ij} = 0$. The task sequencing process continues until all tasks are scheduled and processed according to their deadlines using MSCMCC. All mobile tasks are passed through multiple components to complete their executions.

Algorithm 1 is defined to schedule tasks on each virtual machine using heterogeneous MCC servers. The goal is to minimize R_c by scheduling and sequencing tasks according to the given parameters.

3.8. MCC server resource attaining

The proposed system aims to optimize costs and involves using heterogeneous MCC servers. The ideal approach is to choose the most effective edge MCC server for processing all extended sequential tasks to achieve the best results. Our task scheduling objective for microservices applications is to minimize processing and computation costs for the MCC. One of the challenges is selecting an MCC server with the minimum cost δ_j. This can be achieved using Eq. (26) and Eq. (27), which demonstrate the unit cost δ_j and smaller MCC server costs, respectively.

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

Eq. (26) utilizes the ρ_j to determine the size of the MCC server M_j based on its processing power, memory, and task processing capacity. The resulting output of Eq. (26) is the overall cost associated with the chosen MCC server. Eq. (27) determines the unit cost δ_j, which considers the demands for MCC tasks, MCC VM, and MCC bandwidth handling. The unit cost is computed to facilitate the offloading of tasks with a cost-effective approach.

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

Once the servers have successfully processed the initial level tasks, the leftover resources on the MCC server M_i are not required to be wasted and are represented by the average of μ_{M_i}. The dot product γ_i denotes the task that maximizes the products and their primary operations. The values of γ_i can be determined using Eqs. (28)–(31), which comprehensively explain the proposed system.

$$\gamma_i = \overline{\mu_{M_i}} \, {}^{*}\overline{\beta_{\tau_{ij}}} \tag{28}$$

$$\gamma_i = S_{ci} \, {}^{*}R_j^l {}^{*}q_i + q_i \, {}^{*}v_j^l + c {}^{*}b_j^l \, {}^{*}q_i \tag{29}$$

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

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

The symbol β_(t_(*j)) represents the scheduling of resource management tasks on the MCC server M_j, which has varying types and quantities of resources. The goal is to match tasks with the optimal and most suitable resources in the heterogeneous MCC server system. Tasks are characterized by vector attributes like the deadline, data size, and workload, while resources have vector attributes like bandwidth, cost, VM capacity, and storage. To accomplish this resource matching, we utilize two algorithms: The technique for Order of Preference by Similarity to the Ideal Solution (TOSS) model and the Analytic hierarchy processing (AHP) model. Our approach uses Algorithm 2 to match MCC server resources to tasks, where resources and tasks are inputted sequentially. The resulting frequent list array, FLIST[], stores fulfilled task requirements that are then allocated to the MCC server MC_j.

Algorithm 2: Resource Matching on MCC Server

Input: G= $\{g_1, g_2, \ldots, g_n\}$, $T = \{t_1, t_2, \ldots\ldots, t_n\}$, $t_i \leftarrow \{S_{cj}, T_w, T_d\}$, $k_j \leftarrow \{R_j^l, v_j^l, v_j^l\}$

Output: *FLIST*[]

Steps

1. Start
2. while(G) do
3. while(T) do
4. if($t_i \equiv k_j$) *then*
5. $AHP(R_j^l, v_j^l, v_j^l)$;
6. $TOSS(S_{cj}, T_w, T_d)$;
7. $add(PLIST[k_j, t_i])$;
8. else
9. *goto Step* 4;
10. store($PLIST(k_j, t_i)$;
11. Return $PLIST[k_j, t_i]$;
12. End Loop;
13. End;

Step 4 of Algorithm 2 checks the resource requirements of all incoming tasks t_i for the MCC server. If the resources match the requirements of the MCC server, the algorithm returns true. Then, it returns false. Once a true result is obtained, the matching list is added to the Frequent List PLIST [k_j, t_i]. This process is repeated for all possible tasks received from mobile devices to match the MCC server's heterogeneous requirements.

3.9. MCC server task sequencing

The tasks arrive randomly from various mobile devices, and their arrival rate is tracked through the Poisson Process. The task sequences have no specific order; they are allocated randomly without any rule. All tasks must be sequenced, and the sequenced tasks should be properly formatted and provided in the correct sequential order. Three parameters are considered to sequence the tasks: task size, deadline, and slack time. Four rules have been developed and deployed in the proposed system to sort all incoming unordered tasks.-

1. The First Come First Served (FCFS) algorithm involves arranging tasks based on their arrival times. The tasks are sorted and placed in a queue in the order they arrive, without any specific priority due to the nature of FCFS. Even late tasks are arranged based on their arrival times. The effectiveness of FCFS in addressing task lateness is determined by its adherence to this sorting approach.

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

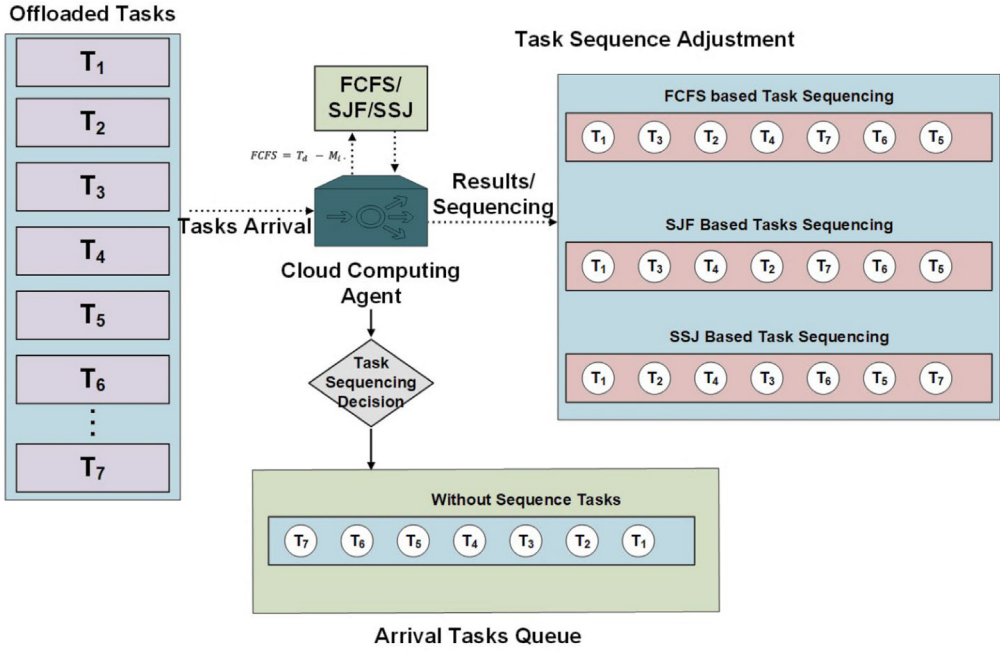

Fig. 3. The Cloud Computing Agent utilizes FCFS, SJF, and SSJ task scheduling algorithms to adjust task sequences for applications based on their arrival time.

2. The SJF (Shortest Job First) scheduling algorithm arranges tasks based on their computation time. The shortest tasks are prioritized for execution ahead of longer tasks, resulting in minimized lateness for the shorter tasks. The longer tasks are scheduled for execution later.
3. The SSF (Shortest Size First) approach, as outlined in Ref. [34], involves sorting tasks based on their size, with shorter tasks given priority over longer ones. This method uses pre-emptive task scheduling for task sequences, resulting in efficient and dependable data provision.

The tasks offloaded from mobile devices arrived at the MCC in a random order, and initially, they were arranged using a first-come, first-served (FCFS) sequence. The FCFS method arranges the tasks cost-efficiently, following the order sequence rules. Fig. 3 illustrates the different task sequencing techniques available for task offloading, including FCFS, SJF, and SSF, each producing different scheduling and sequencing results. The cloud computing agent chooses the best method to achieve optimal task sequencing based on the problem objectives. The cloud computing agent selects the scheduling algorithm and computes the task values based on the task size, deadline, and slack time to decide which tasks to prioritize. Once the task values are calculated, they are matched with the computational capabilities of FCFS, SJF, and SSF, as explained in Refs. [30], [31], and [32], respectively. Since tasks are processed in batches, the cloud computing agent selects the best sequencing method (FCFS, SJF, or SSJ) for each batch of incoming tasks based on decision parameters that determine whether the tasks are computed based on their sequence numbers. Fig. 3 provides an overview of the offloaded tasks, the responsibilities of the cloud computing agent, and the task sequencing adjustment for FCFS, SJF, and SSJ tasks.

3.10. Task scheduling

Until now, we have completed the resource matching and task sequencing. We need to adopt a task-scheduling methodology to address the problem. However, the initial implementation of the task scheduling policies described in the paper is insufficient for calculating the mobile application's cost. The cost can be evaluated in various ways, and due to concurrent changes in the MCC cloud network, the initial selection may not be appropriate for task scheduling. Furthermore, the instability of MCC resources is another reason not to finalize the initial solution. Therefore, a new and improved solution is necessary to enhance task scheduling. For instance, we can consider executing tasks T_1 and T_2 on

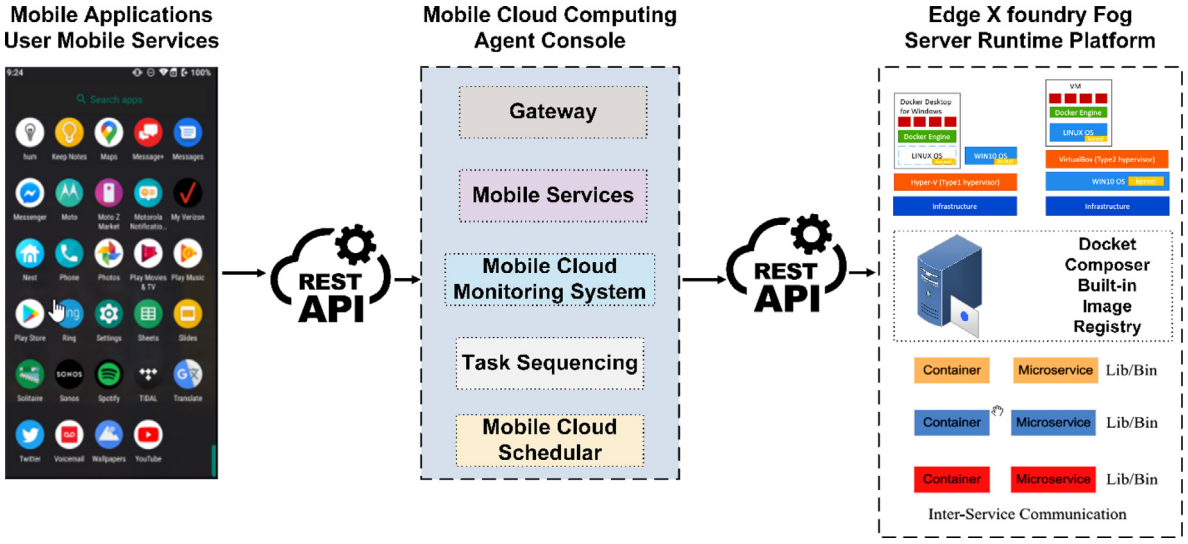

Fig. 4. The task offloading for MSCMCC is done through the Mobile Cloud Computing Agent Console using the REST API. Afterward, the tasks are transmitted to the Edge X Foundry Fog Server Runtime Platform, which is processed on the MCC VM server.

heterogeneous MCC servers with VMs K_1 and K_2. The required resources for T_1 and T_2 include a deadline of 24 and 50, data sizes of 15 Mb and 35 Mb, and CPU requirements of 12 and 32, respectively.

At first, MCC server K_1 is assigned task T_1, while MCC server K_2 is assigned task T_2. The overall cost of the application is the combined sum of the costs for both MCC servers. Eq. (33) illustrates the total cost of the application, indicating the reduced cost necessary for the application to offload the task.

$$\omega_{\text{total}} = \sum_{i=1}^{2} K_i * T_i \tag{33}$$

If all tasks are scheduled on the MCC server K_2, the total cost of the application is determined solely by the cost of the MCC server K_2. Although the tasks are executed on the same server, this reduces the computational costs of the application. Fig. 4 illustrates multiple solutions for sequencing a single task, and we are willing to accept the worst solution for the processes. The scheduler's challenge is to select the best solution that reduces the system's total internal cost. In the example, the scheduler chooses an MCC server with resources that match the cost in the resource list, but additional optimization is necessary to achieve high-cost reduction. Initially, the MCC server had a high cost of available resources, so the main challenge was to reduce the resource utilization cost while utilizing the maximum resources. Therefore, we propose an improved Task Scheduling methodology that significantly improves the resource utilization of MCC servers. The primary objective of the scheduling is to assign tasks to MCC servers with the lowest cost during the initial scheduling phase, which is the only way the Task Scheduling Algorithm can enhance the MCC server's performance. The scheduler eliminated extra and expensive costs during the initial collaboration stage. However, we also propose task scheduling algorithms that enhance scheduling performance and solve the resource optimization problem on the MCC Server. Algorithm 3 defines task scheduling on the MCC server with optimized resource utilization. It takes as input a set of tasks that must be scheduled on a heterogeneous MCC server. The algorithm executes as follows:

- Online 1–6 involves the declaration of variables. MCC servers are then sorted in descending order based on the values of ρ_j and γ_i (derived from Eqs. (27) and (29)).
- Following this, the MCC server is loaded with all applications, and their respective resource and task requirements are specified on lines 7 to 10.

Algorithm 3: Task Scheduling Algorithm

Input: G= $\{g_1, g_2, \ldots, g_n\}$, $T = \{t_1, t_2, \ldots\ldots, t_n\}$, $M_i \leftarrow \{M_1, M_2, M_3, \ldots\ldots, M_n\}$

Output: S_{ij} $(Task\ Scheudling), \gamma_j$ $(MCC\ Server\ State)$

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. $Vector\ Attribute \rightarrow Y_{M_j}$;
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 → δ_j (smallest MCC cost)
14. $t_i \leftarrow M_j$;
15. $h_t^R \equiv M_j$;
16. $t_i\ (largest\ size) \rightarrow M_j$
17. $T_i \leftarrow T_i\ |\{t_i\}|$;
18. $Estalish\ \{S_{ij}, \gamma_j\} = 1$;
19. $J_j \leftarrow J_j \cup \{M_j\}$;
20. $min\ R_c \leftarrow S_{ij}$; : 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. $swap(t_i(M_{g1})\ to\ t_i(M_{g2}))$
25. $assign(S_{i(g1)} = 1)$
26. $(min\ R_c) * \leftarrow S_{ij}$: 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. $End - Loops$;

- The task involves selecting the smallest MCC server δ_j from a set of T tasks, including some unpublished tasks G. The unscheduled tasks are assigned to the MCC server based on their resource requirements and available resources the server. Then, the smallest unit cost server M_j is selected from the available MCC servers M_i. If the scheduled cost of the available fog servers meets the resource demand, the most significant task is scheduled on the MCC server M_i. However, if the server cannot meet the requirements, it is selected to fulfill the demand. Steps 11–20 outline these procedures.
- We select MCC servers M_{g1} and M_{g2} and apply the cost function W. Swapping a task between the MCC servers is only allowed if the resources on M_{g1} fulfill the requirements of task t_i on M_{g2}. The task scheduling variable executes the task swapping on both servers. Once all the tasks on M_{g2} are completed, we update its state and choose a new server from E with the lowest cost to efficiently propose new information.

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 h
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 $\approx$ 1200 Mbps
12	$B_{MCC}^{wj(down)}$	1200 $\approx$ 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

3.11. Time complexity of TSMCO

The TSMCO comprises various components such as Resource Matching, Task Sequences, and Task Scheduling. These components are analyzed individually, and their combined use is optimized to determine the time complexity. Firstly, Resource Matching involves utilizing the TOPSIS and AHP methods for task matching on heterogeneous servers. The time complexity for Resource Matching is calculated as $O(M \times T)$, where M is the MCC server resources for multi-criteria, and T is the number of tasks arranged in pair-wise matching. Secondly, Task Sequencing involves sorting all tasks based on shortest size, deadlines, and lateness, using the $O(m \log n)$ method where N is the number of sorted tasks and M is the employed method for task sorting. Finally, Task Scheduling involves scheduling all MCC servers based on descending order of δ_j and C_j. The time complexity for Task Scheduling is measured as $O(\log M)$. $O(\log M) + N$ for all MCC servers according to their descending order of price and load. Here, N represents the task-swapping process in the time complexity of different MCC servers.

4. Performance evaluation

We have generated practical results from various simulators for mobile and microservices-based applications to assess the effectiveness of the TSMCO and MSCMCC frameworks. Table 4 describes the simulation parameters.

The process has been broken down into several parts based on the simulation parameters specified in Table 4. These parts include the implementation of MSCMCC, calibration of metric parameters and components, comparison of the TSMCO offloading framework, and the comparison of algorithms and task scheduling. Resource specifications for the MCC servers are listed in Table 5, while Table 6 analyzes the workload for mobile applications.

4.1. Comparison framework and approaches

The results are compared to the existing approach based on primary considerations. Fig. 4 illustrates the implementations of MSCMCC via Mobile Application Services, Mobile Cloud Computing Agent Console, and Edge Mobile Server Runtime Platform.

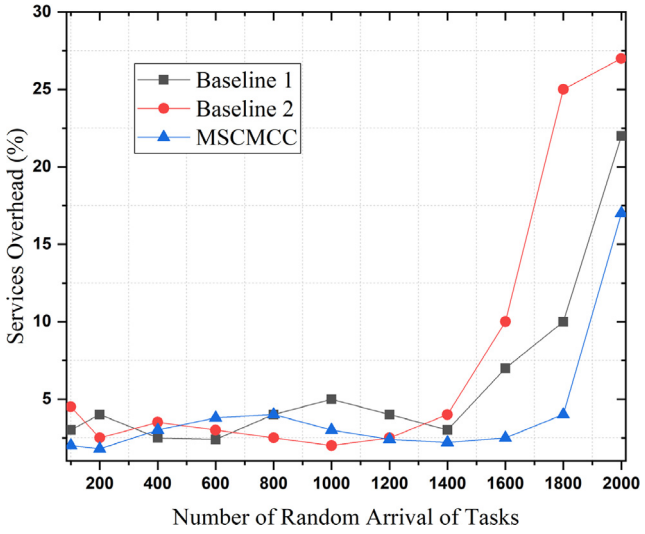

Fig. 5. The time it takes for microservices to start up.

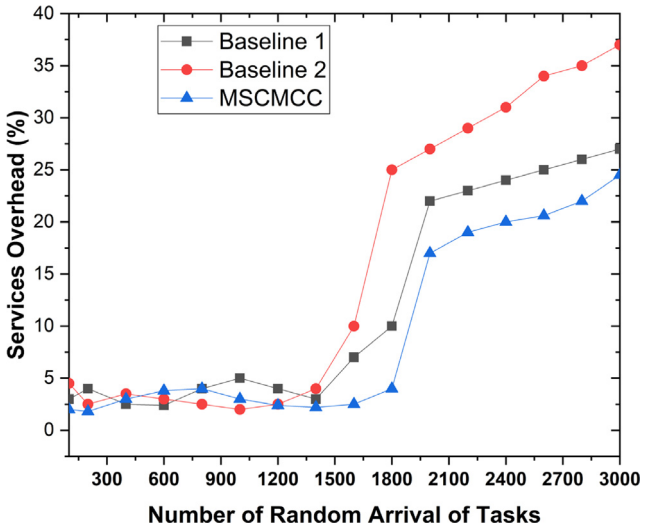

Fig. 6. A scheme for partially offloading 3000 tasks in the proposed work.

- We have developed a VM-based offloading framework for scheduling MCC tasks compared with existing frameworks in [35–38]. We aim to offload the entire mobile application, including all microservices, to MCC cloud servers. Baseline-1 compares the results of our proposed solution, MSCMCC and TSMCO, regarding microservices boot time, task arrival time, CPU utilization, cost, deadline computations, and fault rate. We have implemented the techniques in [39–42] with similar results, but our proposed technique improves these results through practical simulation-based testing.
- Baseline-2/Hypothesis-2 involved utilizing virtual machines to develop a dynamic computational offloading-based framework. This framework was compared to strategies detailed in [34,43,44] to evaluate and contrast their performance. The study's goal was to offload complete mobile applications to heterogeneous servers based on the availability of resources. The obtained results were then compared to existing methodologies. Fig. 4 depicts the MSCMCC framework and TSMCO framework and provides a comprehensive explanation of the proposed system. Additionally, Figs. 5, 6, 7, 9, 10, 11, and 12 were used to support the implementation of the proposed system.
- Hypothesis-3/Baseline-3 involves comparing the cost and task deadline estimation with an additional baseline-3 chosen from the methodology implemented in [45]. The cost function is implemented using the CCCOF framework. The findings reveal that the cost of the existing methodology is less than that of the cost model presented in Fig. 9.

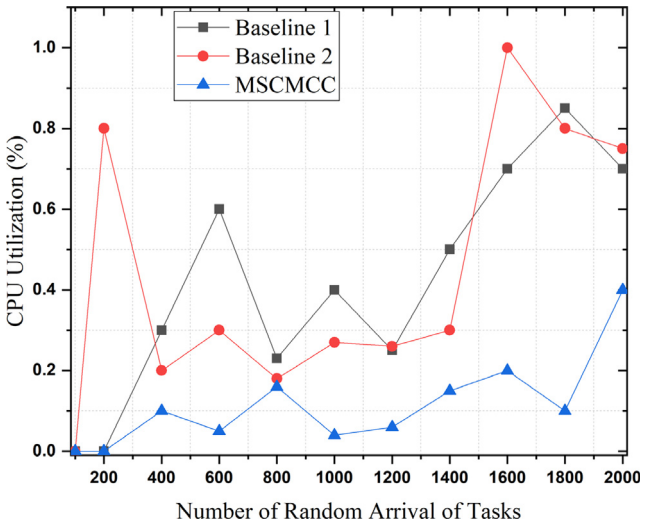

Fig. 7. Resource utilization of the CPU.

4.2. Performance metrics

This paper presents an implementation plan for experimental results recommended by the collaborative efforts of various components. Table 4 outlines the application tasks generated from this collaborative process. The experiment involves five distinct applications, each with its type considered for analysis. To ensure timely execution, every task has a set deadline. The task deadlines in this experimental setup are determined using the following equation.

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

To determine the early completion time and the deadline for the final task interval, it is necessary to establish the deadline for the task $T_d^{a,i}$. The degree of stringency in the task deadline, denoted by γ, is indicated by five values: 0.2, 0.4, 0.6, 0.8, and 1, resulting in five distinct task deadlines (D1, D2, D3, D4, and D5) for each task on a mobile device. The algorithm's performance is evaluated using performance metrics verified through Eq. (31). The Relative Percentage Division (RPD) statistical analysis method is employed to compute the performance division technique for MTOP, VFCN, and CTOS. The paper's findings effectively assess the energy consumption of different devices provide an algorithm for evaluating computational throughput using efficient parameters. Eq. (34) defines the RPD estimation for the proposed method.

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

where, P_a shows the objective function.

4.3. MSCMCC implementation

Defining the early finished time and task deadline is necessary for $T_d^{a,i}$ task. Our implementation of a Mobile Cloud-based application on mobile devices involved using Android Studio, specifically the Huawei Y9 2019 Mobile Model emulator, as shown in Fig. 5. We assessed the Edge X Foundry using an open-source platform. We built the MSCMCC framework, comprising three main components: the Mobile Users layer, Mobile Cloud Agent Control layer, and Mobile Cloud Resources Layer. Mobile applications send their related tasks to the Mobile Cloud Computing Agent Console using REST API. The JSON format interprets requests and responses from the Mobile Cloud Computing Agent via a Gateway Interface. The console interface processes the API request promptly and determines the type of services required to execute the offloaded task based on its characteristics. The Mobile Cloud Monitoring System monitors the task list and system stability. At the same time, Task sequencing orders the tasks logically, and Task Scheduler schedules tasks to heterogeneous Mobile Cloud servers for execution. The Run Time is a complete system environment based on the system scenario. The Java Runtime Virtual Machine (JVM) runs the Java program efficiently, like Windows Docker Virtual Machine. Autonomous Microservices are created using containers registered with a Mobile Cloud server through registry services to consume services efficiently. Using REST API for inter-services communication among microservices minimizes overhead.

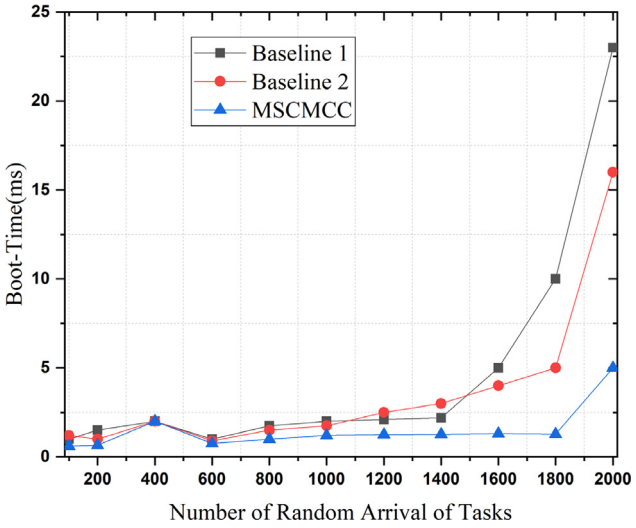

Fig. 8. The microservices overhead.

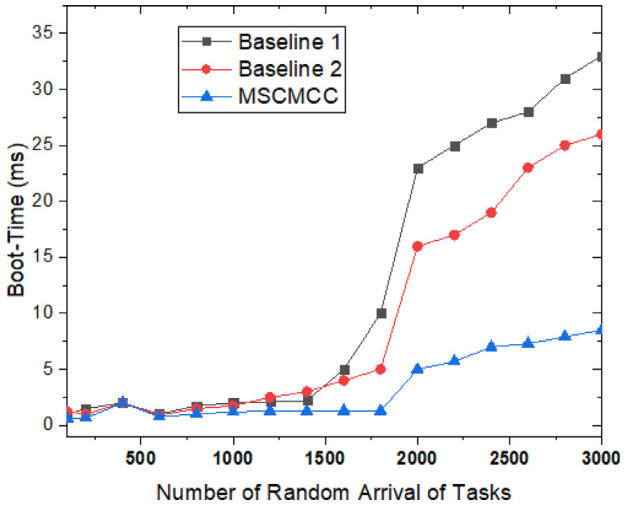

Fig. 9. The impact of Microservices on boot time, measured in milliseconds, and its correlation with the random arrival of 3000 tasks.

4.4. Comparison of offloading frameworks

The Microservices Container-Based Mobile Cloud Computing (MSCMCC) framework proposed in this study demonstrates superior performance compared to existing techniques. The framework reduces bootup time and improves resource utilization due to the utilization of lightweight VMs instead of heavyweight ones. The study shows through simulation and experimental results that the MSCMCC technique effectively enhances the overhead of the service for the arrival of tasks in percentage value. In addition, the proposed approach exhibits less time for microservices-based applications, less CPU usage by around 22%, and less boot time than existing techniques by 17%. The results indicate that the MSCMCC framework effectively improves overhead, bootup time, and resource utilization. The study finds that the lightweight VM utilization during scheduling makes the proposed task scheduling framework efficient for delay-sensitive mobile applications. Furthermore, the proposed system minimizes the cost of healthcare applications by 25%, augmented reality by 23%, E-Transport tasks by 21%, and 3-D games tasks by 19%. On average, the boot time of microservices applications is reduced by 17%, resource utilization by 36%, and task arrival time by 16%.

4.5. Task sequencing

Task sequencing rules, including FCFS, SJF, and SSJ, are utilized to arrange tasks in sequential order for scheduling. These rules work in Fig. 8(a), while Fig. 8(b) shows the mean plot of alpha with a 95% HSD interval and the mean plot for random

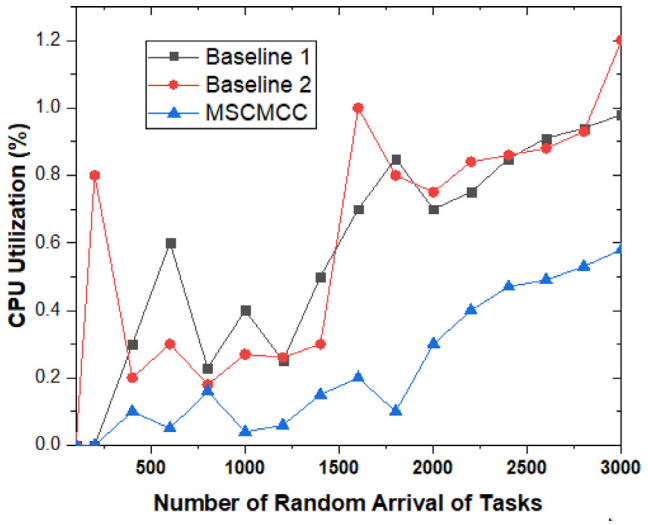

Fig. 10. A scheme that partially offloads processing resources is proposed to optimize CPU utilization when dealing with 3000 randomly arriving tasks.

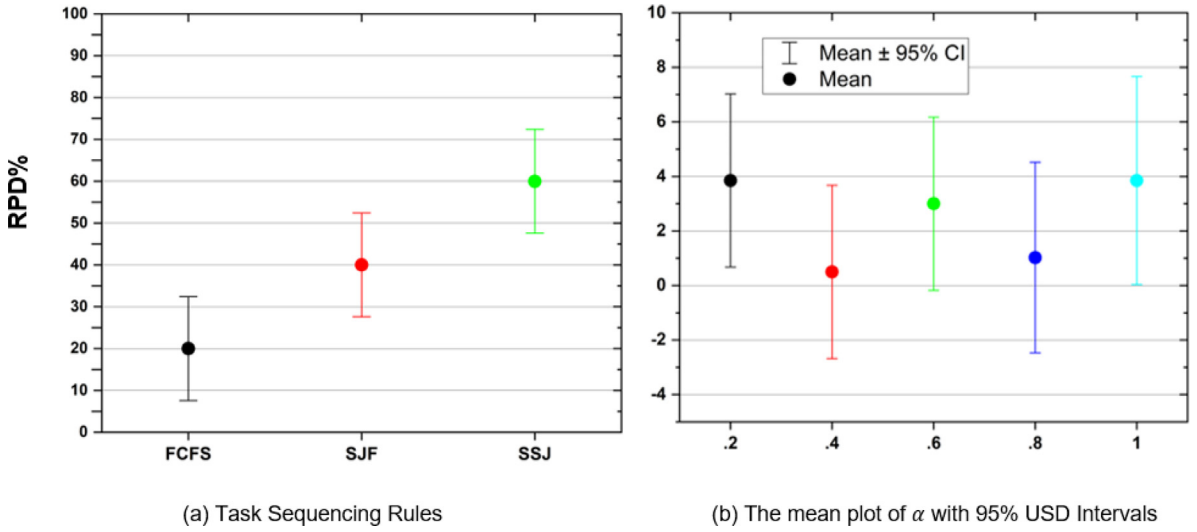

Fig. 11. The average plot for alpha along with a 95% HSD interval and the average plot for the arrival of random mobile tasks with a 95% HSD interval.

mobile tasks arrival with 95% HSD intervals. Our proposed approach effectively compares the results among these rules and extends the results for RPD value in percentage. To implement the results properly, our proposed technique uses SSJ, which effectively improves the results compared to existing methods. The results demonstrate that our approach enhances the proposed technique's working with an effective scenario. Additionally, Fig. 8(b) illustrates that RPD's significance for SSJ is lower than SSTF and SPF for practical elaboration.

4.6. Task scheduling

The efficient management of costs in mobile cloud computing task scheduling is critical for mobile applications. This study considers task deadlines and mobile application costs, specifically computation and communication costs, during task scheduling using mobile cloud servers. This research aims to ensure timely task completion while minimizing mobile application costs. Figs. 9 and 10 demonstrate the TSMCO framework that reduces mobile application execution costs. The proposed scheme and scenario ensure mobile applications are within their deadlines. Fig. 10 displays the deadlines for various tasks, including Healthcare applications, Augmented Reality Tasks, E-Transport Tasks, and 3D-Game Application Tasks. These tasks have soft deadlines that allow for scheduling decisions and fault tolerance. If the initial deadline is

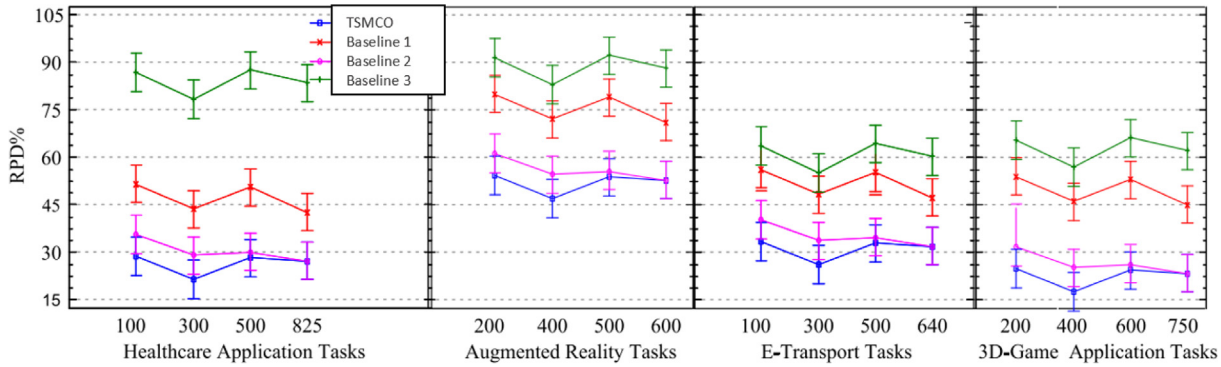

Fig. 12. The RPD% is a factor to be considered in meeting the deadlines for Healthcare, Augmented Reality, E-Transport, and 3-D Game application.

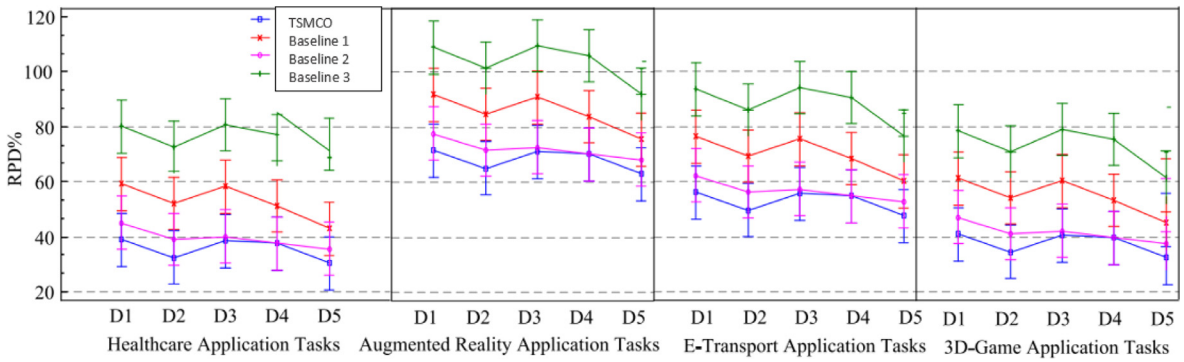

Fig. 13. TSMCO was compared with three other techniques to evaluate the total cost incurred by healthcare applications (100 to 825 tasks), Augmented reality (200 to 600 Tasks), E-Transport (100 to 640 Tasks), and 3D-Games (200 to 750 tasks) in terms of RPD%.

not met due to an MCC server's unavailability, a secondary deadline is set for the task to be offloaded and processed. The proposed system is evaluated by setting up five different deadlines for various tasks, and the iterative features continuously improve the system's performance until it reaches the optimal solution (see Figs. 13–15).

4.7. Task failure ratio

Our task scheduling mechanism demonstrates an improvement in the task failure ratio compared to existing techniques that only utilize basic approaches for task scheduling. Figs. 11 and 12 illustrate the comparison of the task failure ratio between our proposed and existing techniques, indicating that our approach outperforms existing techniques in the dynamic allocation of Mobile Cloud Servers.

The TSMCO framework that has been proposed demonstrates efficient performance in dynamically allocating resources to various situations. It has significantly improved the cost of the system while also providing efficient resource constraints for meeting deadlines. Additionally, the framework effectively executes mobile tasks within the given deadlines and expands the execution cost.

5. Conclusion and future work

In this study, we propose a novel task scheduling approach for a microservices-based mobile cloud computing framework. Our proposed system, MSCMCC, utilizes microservices applications to handle delay-sensitive tasks and includes a mobility-aware framework to reduce application costs. Additionally, we introduce the TSMCO framework, which effectively manages task offloading through three steps: resource provision and matching, task sequencing, and task offloading. Our experimental results demonstrate that MSCMCC and TSMCO effectively utilize the mobile device and MCC server resources while reducing microservices-based application boot time and overhead time. Compared to existing techniques, the proposed method achieves lower overhead time for microservices-based tasks and lower cost for each application in baselines one and two. Moreover, our results highlight the efficient utilization of server resources achieved through MSCMCC and TSMCO. This can reduce mobile server bootup time, minimize microservices latency, and decrease server cost to reduce latency in mobile cloud servers.

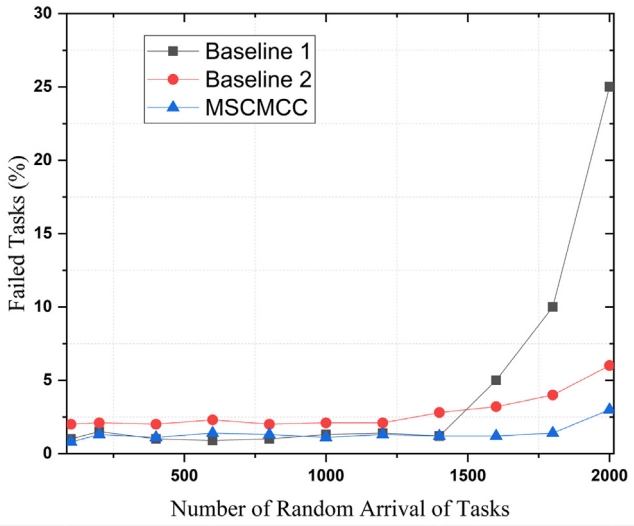

Fig. 14. The ratio of task failures during task scheduling.

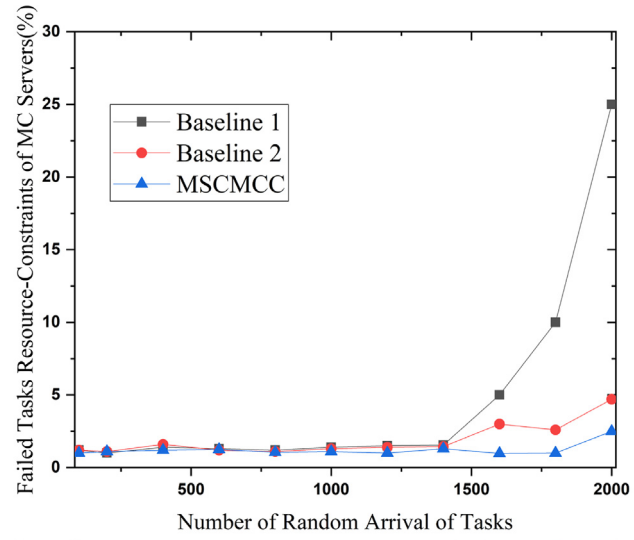

Fig. 15. Mobile cloud servers with limited resources can fail in tasks.

We plan to implement a microservices-based task offloading framework for IoT and mobile applications that prioritizes privacy. This framework will be designed to deploy on mainstream platforms such as Azure, Amazon, and Google and will consider both security and transient failures. Additionally, we aim to incorporate machine learning or ANN for task scheduling decision-making.

Funding statement

The authors are thankful to the Deanship of Scientific Research at Najran University, Saudi Arabia for funding this work under the research code NU/DRP/SERC/12/1.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data that has been used is confidential.

Acknowledgment

The authors extend their appreciation to the Deanship of Scientific Research at Najran University for funding this work, under the Distinguished Research funding Program grant code NU/DRP/SERC/12/1.

References

[1] A. Amini Motlagh, A. Movaghar, A.M. Rahmani, Task scheduling mechanisms in cloud computing: A systematic review, 33 (6), 2020, e4302.
[2] N. Parajuli, A. Alsadoon, P. Prasad, R.S. Ali, O.H.J.M.T. Alsadoon, Applications, A recent review and a taxonomy for multimedia application in mobile cloud computing based energy efficient transmission, 79 (41), 2020, pp. 31567–31594.
[3] V.K. Kaliappan, S. Gnanamurthy, C.S. Kumar, R. Thangaraj, K.J.M.T.P. Mohanasundaram, Reduced power consumption by resource scheduling in mobile cloud using optimized neural network, 2021.
[4] A. Aliyu, et al., Mobile cloud computing: taxonomy and challenges, 2020, 2020.
[5] S. Guo, B. Xiao, Y. Yang, Y. Yang, Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing, in: IEEE INFOCOM 2016-the 35th Annual IEEE International Conference on Computer Communications, IEEE, 2016, pp. 1–9.
[6] X. Wang, K. Wang, S. Wu, S. Di, K. Yang, H. Jin, Dynamic resource scheduling in cloud radio access network with mobile cloud computing, in: 2016 IEEE/ACM 24th International Symposium on Quality of Service, IWQoS, IEEE, 2016, pp. 1–6.
[7] M. Chen, S. Guo, K. Liu, X. Liao, B. Xiao, Robust computation offloading and resource scheduling in cloudlet-based mobile cloud computing, 20 (5), 2020, pp. 2025–2040.
[8] X. Liu, Y. Li, H.-H. Chen, Wireless resource scheduling based on backoff for multiuser multiservice mobile cloud computing, 65 (11), 2016, pp. 9247–9259.
[9] P. Akki, V.J.W.P.C. Vijayarajan, Energy efficient resource scheduling using optimization based neural network in mobile cloud computing, 114 (2), 2020.
[10] J. Zare, S. Abolfazli, M. Shojafar, A. Kamsin, Resource scheduling in mobile cloud computing: taxonomy and open challenges, in: 2015 IEEE International Conference on Data Science and Data Intensive Systems, IEEE, 2015, pp. 594–603.
[11] E. Ahmed, A. Gani, M. Sookhak, S.H. Ab Hamid, F. Xia, Application optimization in mobile cloud computing: Motivation, taxonomies, and open challenges, 52, 2015, pp. 52–68.
[12] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, A. Patti, Clonecloud: elastic execution between mobile device and cloud, in: Proceedings of the Sixth Conference on Computer Systems, 2011, pp. 301–314.
[13] E. Cuervo, et al., Maui: making smartphones last longer with code offload, in: Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, 2010, pp. 49–62.
[14] S. Kosta, A. Aucinas, P. Hui, R. Mortier, X. Zhang, Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading, in: 2012 Proceedings IEEE Infocom, IEEE, 2012, pp. 945–953.
[15] A.A. Heidari, S. Mirjalili, H. Faris, I. Aljarah, M. Mafarja, H.J. Chen, Harris hawks optimization: Algorithm and applications, 97, 2019, pp. 849–872.
[16] I. Attiya, M. Abd Elaziz, S.J.C. i. Xiong, neuroscience, Job scheduling in cloud computing using a modified harris hawks optimization and simulated annealing algorithm, 2020, 2020.
[17] M. Satyanarayanan, P. Bahl, R. Caceres, N. Davies, The case for vm-based cloudlets in mobile computing, 8 (4), 2009, pp. 14–23.
[18] S. Goundar, A. Bhardwaj, S. Chand, Control, and communications, task offloading concept using cloud simulations in mobile computing, 12 (3), 2021, pp. 243–263.
[19] M.A. Sahito, A. Kehar, Dynamic content enabled microservice for business applications in distributed cloudlet cloud network, 9 (7), 2021.
[20] A. Lakhan, Q.-U.-A. Mastoi, M. Elhoseny, M.S. Memon, M.A. Mohammed, Deep neural network-based application partitioning and scheduling for hospitals and medical enterprises using IoT assisted mobile fog cloud, 2021, pp. 1–23.
[21] X. Ma, A. Zhou, S. Zhang, Q. Li, A.X. Liu, S. Wang, Dynamic task scheduling in cloud-assisted mobile edge computing, 2021.
[22] A.M. Rahmani, et al., Towards data and computation offloading in mobile cloud computing: Taxonomy, overview, and future directions, 2021, pp. 1–39.
[23] R.B. Mulinti, M.J. P.-t. P. N. Nagendra, Applications, An efficient latency aware resource provisioning in cloud assisted mobile edge framework, 14 (3), 2021, pp. 1044–1057.
[24] A. Lakhan, et al., Cost-efficient service selection and execution and blockchain-enabled serverless network for internet of medical things, 18 (6), 2021, pp. 7344–7362.
[25] A. Ali, et al., An efficient dynamic-decision based task scheduler for task offloading optimization and energy management in mobile cloud computing, 21 (13), 2021, p. 4527.
[26] X. Ma, H. Xu, H. Gao, M. Bian, Real-time multiple-workflow scheduling in cloud environment, 2021.
[27] A. Lakhan, M.A. Dootio, T.M. Groenli, A.H. Sodhro, M.S.J.E. Khokhar, Multi-layer latency aware workload assignment of e-transport iot applications in mobile sensors cloudlet cloud networks, 10 (14), 2021, p. 1719.
[28] N. Chaurasia, M. Kumar, R. Chaudhry, O.P. Verma, Comprehensive survey on energy-aware server consolidation techniques in cloud computing, 2021, pp. 1–56.
[29] L. Abualigah, A. Diabat, P. Sumari, A.H. Gandomi, Applications, deployments, and integration of internet of drones (IoD): a review, 2021.
[30] A. Samanta, Z.J. Chang, Adaptive service offloading for revenue maximization in mobile edge computing with delay-constraint, 6 (2), 2019, pp. 3864–3872.
[31] A. Samanta, J. Tang, Dyme: Dynamic microservice scheduling in edge computing enabled IoT, 7 (7), 2020, pp. 6164–6174.
[32] A. Samanta, F. Esposito, T.G. Nguyen, Fault-tolerant mechanism for edge-based IoT networks with demand uncertainty, 8 (23), 2021, pp. 16963–16971.
[33] C. Tang, M. Hao, X. Wei, W. Chen, Energy-aware task scheduling in mobile cloud computing, Distrib. Parallel Databases 36 (3) (2018) 529–553.
[34] F.J.D.S. Yang, Applications, Cloud computing virtual resource dynamic system allocation and application based on system architecture, 30 (5), 2021, pp. 753–770.
[35] A. Lakhan, D.K. Sajnani, M. Tahir, M. Aamir, R. Lodhi, Delay sensitive application partitioning and task scheduling in mobile edge cloud prototyping, in: International Conference on 5G for Ubiquitous Connectivity, Springer, 2018, pp. 59–80.
[36] C. Tang, M. Hao, X. Wei, W.J.D. Chen, P. Databases, Energy-aware task scheduling in mobile cloud computing, 36 (3), 2018, pp. 529–553.

[37] X. Ma, S. Zhang, W. Li, P. Zhang, C. Lin, X. Shen, Cost-efficient workload scheduling in cloud assisted mobile edge computing, in: 2017 IEEE/ACM 25th International Symposium on Quality of Service, IWQoS, IEEE, 2017, pp. 1–10.
[38] S. Wu, C. Niu, J. Rao, H. Jin, X. Dai, Container-based cloud platform for mobile computation offloading, in: 2017 IEEE International Parallel and Distributed Processing Symposium, IPDPS, IEEE, 2017, pp. 123–132.
[39] J. Lee, J. Gil, Adaptive fault-tolerant scheduling strategies for mobile cloud computing, J. Supercomput. 75 (8) (2019) 4472–4488.
[40] D.N. Raju, V. Saritha, Architecture for fault tolerance in mobile cloud computing using disease resistance approach, Int. J. Commun. Netw. Inf. Secur. 8 (2) (2016) 112.
[41] S.K. Abd, S.A.R. Al-Haddad, F. Hashim, A.B. Abdullah, S. Yussof, Energy-aware fault tolerant task offloading of mobile cloud computing, in: 2017 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, MobileCloud, IEEE, 2017, pp. 161–164.
[42] J. Park, H. Yu, H. Kim, E. Lee, Dynamic group-based fault tolerance technique for reliable resource management in mobile cloud computing, Concurr. Comput.: Pract. Exper. 28 (10) (2016) 2756–2769.
[43] R. Tiwari, R. Sille, N. Salankar, P. Singh, Utilization and energy consumption optimization for cloud computing environment, in: Cyber Security and Digital Forensics, Springer, 2022, pp. 609–619.
[44] N.K. Mishra, P. Himthani, G.P. Dubey, Priority-based shortest job first broker policy for cloud computing environments, in: Proceedings of International Conference on Communication and Computational Technologies, Springer, 2021, pp. 279–290.
[45] B. Keshanchi, A. Souri, N.J. 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. 124 (2017) 1–21.