Journal of Big Data

RESEARCH

Open Access

Efficient microservices offloading for cost optimization in diverse MEC cloud networks

Abdul Rasheed Mahesar[1*], Xiaoping Li[1*] and Dileep Kumar Sajnani[1]

*Correspondence:
abdulrasheed@seu.edu.cn;
xpli@seu.edu.cn

[1] School of Computer Science and Engineering, Southeast University, Nanjing 211189, Jiangsu, China

Abstract

In recent years, mobile applications have proliferated across domains such as E-banking, Augmented Reality, E-Transportation, and E-Healthcare. These applications are often built using microservices, an architectural style where the application is composed of independently deployable services focusing on specific functionalities. Mobile devices cannot process these microservices locally, so traditionally, cloud-based frameworks using cost-efficient Virtual Machines (VMs) and edge servers have been used to offload these tasks. However, cloud frameworks suffer from extended boot times and high transmission overhead, while edge servers have limited computational resources. To overcome these challenges, this study introduces a Microservices Container-Based Mobile Edge Cloud Computing (MCBMEC) environment and proposes an innovative framework, Optimization Task Scheduling and Computational Offloading with Cost Awareness (OTSCOCA). This framework addresses Resource Matching, Task Sequencing, and Task Scheduling to enhance server utilization, reduce service latency, and improve service bootup times. Empirical results validate the efficacy of MCBMEC and OTSCOCA, demonstrating significant improvements in server efficiency, reduced service latency, faster service bootup times, and notable cost savings. These outcomes underscore the pivotal role of these methodologies in advancing mobile edge computing applications amidst the challenges of edge server limitations and traditional cloud-based approaches.

Keywords: Mobile edge computing, Cloud, Task scheduling, Microservices, Optimization, Container

Introduction

Recently, the proliferation of interconnected gadgets like smartphones, tablets, and fitness bands has profoundly impacted modern lifestyles, driving the deployment of numerous interconnected devices strategically positioned within specific geographic regions for data detection, transfer, and analysis [1, 2]. This surge is fueled by the rapid expansion of mobile applications such as augmented reality, e-commerce, gaming, healthcare, and social media, collectively contributing to a significant upsurge in data generation typically constrained by deadlines. By 2025, global social media users are projected to exceed 4.41 billion [3]. These applications rely on extensive computational offloading and demand low-latency processing to manage and derive insights from the massive influx of data [4]. Task offloading in mobile edge cloud environments intricately manages mobile devices, edge servers, and the public cloud, optimizing computational efficiency and user experience. Mobile devices, constrained by processing power and battery life, benefit from offloading tasks to nearby edge servers strategically placed near end-users, enabling low-latency processing crucial for applications like augmented reality and IoT analytics [5, 6]. However, edge servers face challenges such as finite capacity. Conversely, the public cloud offers scalable resources and global reach, providing additional computational power and storage flexibility but potentially introducing latency and increased operational costs. Balancing these layers is essential to optimize resource utilization, enhance application performance, and establish robust frameworks capable of meeting diverse computational demands in mobile edge cloud computing [7–10].

In literature, several offloading frameworks and algorithms have been proposed to tackle the issue of task scheduling, computation offloading, and cost minimization [11–15]. However, prior studies employed virtual machines (VMs) as the hosting infrastructure for the mobile operating system in a cloud environment, utilizing hardware virtualization, such as VirtualBox integrated Android X86. One of the main challenges of using VMs in a MEC environment is the significant amount of time it takes to set up the service and the added burden of virtualization when sharing data. However, the applications mentioned above are highly time-sensitive, interactive, and heavily rely on the ability to move effortlessly within the infrastructure of the mobile cloud. Recently, there has been a growing emphasis on utilizing cloud services and operating systems that are container-based. These technologies have garnered significant attention due to their ability to provide faster setup times and lower overhead for operating system virtualization [16, 17]. Constructing a mobile cloud utilizing container-based virtualization is a highly intriguing concept, although it poses numerous obstacles. Developing a mobile runtime using containerization and virtualization techniques has great potential, but also presents several difficulties. Initially [18–21] described applications are divided into components at a detailed level current mobile cloud services of the monolithic type are unable to meet these needs. Furthermore, in the mobile cloud environment, the mobile network conditions, such as bandwidth, latency, and signal strength, undergo intermittent changes.

In this paper, we focus on cost-eficient scheduling in mobile edge cloud computing in heterogeneous environment. The object is to minimize communication and computation cost. Mobile applications consist of independent granular sub-tasks, where each sub-task operates autonomously with its own specific attributes and data. This granularity allows for efficient workload utilization. Each task is characterized by a vector of attributes, including CPU instructions, data size, and execution deadlines. Our consideration of services includes an evaluation based on their cost and performance speed.

Motivation: Smart devices based mobile application increasing progressively. However these application need thin services to minimize delay. Initially, the MEC paradigm relied on heavyweight virtual machines to support user applications, paid for on a pay-as-you-go basis. Consequently, contemporary MEC paradigms face challenges primarily in cost, interaction, and mobility. Moreover, meeting the requirements of Internet of Things applications, such as E-banking (e.g., Transection), demands leveraging a diverse array of services from multiple providers. Hence, efficiently and cost-effectively scheduling resources remains a significant challenge in MEC for mobile applications.

In order to address the aforementioned issues, the study makes the following contributions:

- We presented an innovative mobile edge computing MEC system based on microservices containers MCBMECS, leveraging Docker containers to optimize VM workload and boost performance. The benefits of MCBMECS include minimized service overhead and faster VM boot times.
- study explores various MEC servers, each with its own distinct characteristics." Similarly, different tasks within the application have specific sets of Quality of Service *QoS* requirements. Choosing the right edge server is essential for meeting the requirements of a task efficiently. As a solution to this challenge, we present a resource matching algorithm that aims to align each server with the unique requirements of a given task.
- In our scenario, tasks enter the system in a random manner, following a Poisson process. The offloaded tasks are introduced to the system without any predetermined order, which requires the need for sequencing. To address this, we employ a rule-based task sequencing approach, given the importance of the deadline, size, and slack time in task prioritization, we have established three rules to effectively organize the submitted tasks according to these attributes. There are several rules to consider: Lateness Time First LTF, Earliest Due Date EDD and Shortest Size First SSF.
- Scheduling a collection of tasks across a network of MEC servers can be quite challenging, considering the costs of communication and computation. As a solution to this challenge, we present an affordable task offloading strategy. This scheme efficiently assigns tasks to servers in a systematic manner, aiming to minimize costs and meet deadlines.

The remaining sections of the paper are structured as follows. In Sect. "Related work", we will explore the previous studies that are relevant to the task scheduling problem. Section "Proposed architecture for microservices based-mobile edge computing" provides a detailed explanation and formal analysis of the problem being investigated. Section "Performance evaluation" provides a detailed explanation of the algorithms OTSCOCA, along with their distinct components, are detailed in Sect. "Performance evaluation". The performance evaluation is carried out in Section 5, while Section serves as the conclusion of the study.

Related work

The studies mentioned in refs. [22–26] examined several frameworks and techniques for efficient computation offloading. Their primary objective was to find ways to reduce mobile device power consumption by moving computation to the cloud-either the public cloud or a local edge cloud. The concept of dynamic computation offloading in mobile cloud computing entails making real-time decisions to manage resource limitations. This process involves transferring computationally intensive tasks to an external platform, such as cloud computing or edge cloud, aiming to decrease mobile energy consumption and enhance the performance of mobile cloud applications in terms of response time. Several studies, including those referenced in refs. [27–30] have explored this strategy.The research explored in refs. [4, 27, 31] examines the integration of workflow applications and game theory to facilitate real-time decisions regarding task offloading in mobile cloud computing. These investigations specifically target challenges associated with resource limitations and allocation strategies. The primary objective is to transfer computationally intensive tasks to external platforms like cloud computing or edge clouds. This approach aims to reduce mobile device energy consumption while simultaneously improving the performance metrics of mobile cloud applications, such as response time, makespan, and adherence to deadline constraints.

The framework known as CloneCloud was presented in the paper [32], which was developed with the intention of improving the battery life as well as the execution performance of mobile devices. During the course of the research, task offloading strategies were utilized in order to move compute-intensive workloads to centralized cloud servers. A computational offloading architecture was proposed by ThinkAir [33] that allows smartphone applications to move their workload to the cloud. With the use of mobile phone virtualization in centralized cloud settings the framework makes it possible to offload computation at the method level.The objective of the reference [34] is to investigate the process of offloading tasks from mobile devices to various cloudlets in order to improve the overall cost-effectiveness. In the study lengths of communication and computation are taken into consideration along with the costs associated with meeting deadlines. The works examined in references [35, 36] focus on addressing difficulties associated with enhancing energy efficiency and cost-effectiveness in task scheduling. Their main focus is to improve the lifespan of batteries by transferring computing tasks to cloud infrastructures. The research also seeks to reduce resource expenditures, such as storage and computing charges, by implementing effective job scheduling in various cloud environments.

As the number of mobile devices and apps grows, the practice of computational offloading for mobile applications has become increasingly common and extensively adopted among mobile cloud service customers [37]. This method allows mobile devices with significant computational requirements to offload their tasks to the cloud, where they can be processed on mobile cloud servers. Multiple initiatives have been pursued in various scientific fields to improve the efficiency of both devices and applications [38]. Improving the efficiency of both battery longevity and mobile devices can be accomplished by offloading mobile tasks. The objective of such projects is to shift resource-intensive tasks of applications to mobile cloud support. This aligns with previous studies that aimed to enhance mobile application performance, reduce computational expenses, and utilize the processing capacity of mobile devices [39]. This strategy gives higher importance to factors like energy usage and execution time when scheduling tasks in the given context. presented a pricing scheme for MEC services that focuses on coordinating service caching decisions and controlling task offloading behavior in cellular networks [40]. The goal is to reduce computation and latency costs for all users also made a valuable contribution to this effort, focusing on enhancing user benefits through optimizing service pricing and task offloading dynamics [41]. The detailed comparison between our approach and existing techniques is shown in Table 1.

Based on our best knowledge, efficient microservices offloading can optimize costs in mobile edge cloud networks. The MCBMEC framework is an innovative solution

Table 1 Summary of existing related works

Papers	Research issue	Background	Optimization object	Task failure	Cost	Deadline	Resource type
[4]	Task assignment and migration	Data center energy saving	Energy	No	Yes	Yes	RL-based
[7]	Workload assignment	Mobile edge cloud	Minimize Response Time	No	No	Yes	Vm-based
[30]	Task offloading abd resource allocation	Edge cloud	cost	No	Yes	No	Vm
[29]	Task offloading and scheduling	Cloud computing	cost and energy	Yes	Yes	No	Vm
[27]	Task offloading decision	Vehicular edge computing	cost	No	Yes	Yes	RL-based
[17]	Container scheduling	Edge cluster upgrade	Latency	No	Yes	No	RL-based
Ours	Task offloading and scheduling	Mobile edge cloud computing	Cost	Yes	Yes	Yes	Container based

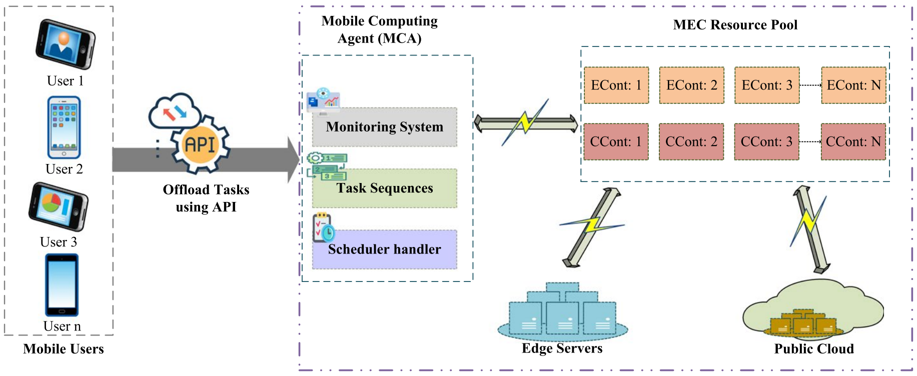

Fig. 1 Proposed MCBMECS system architecture

specifically developed to optimize the execution of application tasks while minimizing costs. Maintaining QoS requires the implementation of the Optimization Task Scheduling and Computational Offloading with Cost Awareness (OTSCOCA) framework. The OTSCOCA mechanism guarantees QoS for applications, reduces resource costs, and ensures that tasks are completed within predetermined time limits.

Proposed architecture for microservices based-mobile edge computing

The proposed architecture, referred to as Microservices Container-Based Mobile Edge Computing (MCBMEC), is illustrated in Fig. 1. The three layers represented in Figure are the Task Scheduling Layer, Mobile Cloud Layer and Mobile Users Layer. In general, tasks initiated by mobile users are intended to be transferred to another device for processing. Based on the varied attributes in upcoming interactions, these tasks are sent over using the API. Once the API tasks have been processed, they are directed to the task scheduling layer, which comprises four primary modules. The function of these modules is to receive the tasks that have been offloaded in the task scheduling layer. The MCA functions as the core component that handles the administration and implementation of all delegated tasks. The MCA engages in collaboration with the components include the system monitor agent, task Sequences and task scheduler handle. MCA, serves as a specialised interface connecting mobile devices and system resources. Its primary function is to collect data from the APIs of mobile devices. This data include various information comprises diverse information, encompassing configuration particulars, measurements and logs. The MEC virtual network consists of several entities, including the system monitoring agent, task scheduling and task sequences which are hosted on MEC servers. The performance evaluation and workload management in MCA are facilitated through the utilisation of three modules.

Task characterization

The mobile application consists of individual tasks that are detailed and specific. In this particular context, the term fine-grained refers to the characteristic of each task having its own data stored within the application. These tasks function autonomously and require various resources for their completion. Each task is defined by a set of vector attributes, which include of the required CPU resources, data size, and a deadline. These attributes are taken into account when scheduling tasks on the fog system. The arrival of these tasks at the MEC system follow a stochastic process. During the process of scheduling, the state of preemption is prohibited.

Resource characterization

Within the context of the cloud paradigm, containers have emerged as a highly efficient approach for virtualizing programmes, particularly for deployment on edge servers. Services refer to a compilation of discrete and autonomous entities that interact with one another via clearly defined APIs. In this study, we examine a heterogeneous edge cloud server that employs virtual machines at the highest tier. Additionally, we leverage the Docker engine in conjunction with MCBMEC, a framework that enables efficient attachment and detachment of microservice containers within the system.

Runtime microservices in MCBMECS

Microservices architecture, often known as an architectural style, organizes an application as a set of independent and self-contained services. Microservices is a software development approach that focuses on the creation of runtime software. The present study examines the variability inherent in the service-oriented architecture paradigm. In the context of a microservices architecture, an application is structured as a sequence of services that exhibit loose coupling. Each individual service is accountable for distinct and detailed functions, and resources are explicitly stated at the service level.

The MEC server in MSBMEC consists of a single virtual machine that may be provisioned on demand. On the other hand, each virtual machine can host several containerized microservices simultaneously. Each microservice, such as the bank account user searching service, represents a business objective and possesses its own set of resources and libraries to execute various tasks. As a result, this approach proves to be efficient for managing detailed tasks of an application when offloading within the system.

Problem formulation

In order to define the task scheduling problem in the Mobile edge computing system, we begin by creating a model for the input tasks that are offloaded by the users, as well as the edge servers that are available. Next, we calculate the resource limitations for the task scheduling problem and introduce a comprehensive optimization framework. The tasks $T = \{t_1, t_2, \ldots t_n\}$ that need to be executed are managed according to a task scheduling strategy, where N represents the total count of input tasks. Each task t_i in the set T is represented by $t_i = \{Wd_i, data_i, dc_i, Sd_i\}$. Here W_{di} represents the computation workload of t_i,D_{ti} denotes the size of the task during transmission,d_{ci} indicates the task deadline, and S_{di} illustrates the task's storage requirements.

In the Mobile edge computing system, we assume the existence of m heterogeneous MEC servers, denoted as $M = \{m_1, m_2, \ldots, N\}$. Each server m_j in the set M can be represented as $m_j = \{B_j^w, \varsigma_j, Sm_j, V_{mec}^j\}$. Here B_j^w, refers to the measurement of the bandwidth linking the centric mobile cloud agent and the mobile cloud server. ς_j denotes the j^{th} MEC server. Sm_j represents the overall storage capacity of the server. V_{mec}^j denotes the quantity of virtual machine Docker instances that have been implemented for microservices, all possessing identical capabilities, within the mec server j. Each Virtual machine V_{mec}^j consists of multiple containers and is responsible for executing numerous microservices to handle offloaded tasks. Every individual microservice has the ability to operate independently, utilising its own set of libraries and database resources while performing tasks.

In addition, let's define b_{ij}^w as the bandwidth needed for task t_i on MEC server m_j when it is scheduled. Given that server m_j state is still active (i.e., $\varphi_j = 1$), the cost C_j represents the resource costs (e.g., RAM, Storage, Bandwidth) incurred during the execution of the offloaded task t_i. Table 2 lists the remaining symbols in detail in

Table 2 Explanation of notations

Notation	Description
N	A set of all task applications
M	Number of MEC Servers
P_{ij}	The Task assign to the MEC server m_j.
φ_j	Represent the MEC server is On or Off
t_i	Represent the ith tasks of application
m_j	The jth MEC server in the system
sd_i	The storage requirement for task t_i
Sm_j	The jth mec cloud server's capacity
wd_i	The computational data or workload associated with a job t_i
B_j^w	Represent the ith tasks of application
dc_i	The task t_i has a deadline constraint
b_{ij}^w	Bandwidth demand increases when a task is scheduled to m_j
V_{mec}^j	The quantity of Docker virtual machines deployed at m_j
ς_j	The computational speed of all virtual machines within the system m_j
C_j	The price of the jth MEC server in the system
$data_i$	Represents Task t_i size in terms of data during transmission

order to save space on the page. By introducing a binary variable p_{ij}, we can indicate whether or not a task t_i is scheduled on mec server m_j.

$$\mathrm{P_{ij}} = \begin{cases} 1, & t_i \leftarrow M_j; \\ 0, & Otherwise \end{cases} \tag{1}$$

Just like the problem of assignment, each task is carefully assigned to a single m_j, Moreover, each MEC server is assigned a single task at a time. The diagram illustrates the allocation of task t_i to mec server m_j.

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

Due to the finite resources available on each server, it is impera+tive to adhere to the following task scheduling constraints. Within each server, it is imperative to have an adequate amount of storage capacity available in order to effectively store the processed tasks. Failure to do so may lead to the unfortunate consequence of data loss pertaining to these tasks. Hence, it is imperative that the cumulative storage demand of every task allocated to server e_j does not surpass the available storage capacity of server e_j.

$$\sum_{i=1}^{M} p_{ij} * Sd_i \le Sm_j \tag{3}$$

Because of resource constraints, MEC server j has a limited number of virtual machine dockers to offer microservices for each task. As a result, the required tasks' workload must not exceed the capability of the deployed virtual machine. This statement is defined as follows:

$$\sum_{i=1}^{N} p_{ij} * wd_i \le V_{mec}^{j}. \tag{4}$$

Efficient servers are assigned to each offloaded task j. This is decided upon by the MEC agent, who in turn decides the scheduling of task t_i. This is how we rank the job completion on server j

$$T_e^{t_i} = \sum_{j=1}^{M} P_{ij} * \frac{wd_i}{\varsigma_j^{mec}} \tag{5}$$

When the processing of task t_i is transferred to the MEC server, there is extra communication involved in this process, and then the task is returned by the MEC server.

$$RTT = \left(\frac{data_{in}^{i}}{B_{mec}^{up}} + \frac{data_{out}^{j}}{B_{mec}^{down}} \right) \tag{6}$$

The size of a task t_i is indicated by $data_{in}^{i}$, whereas $data_{out}^{j}$ represents the output of The task t_i is processed on server j. Furthermore, B_{mec}^{up} and B_{mec}^{down} represent the rates at which data is transmitted from the application to server j (uplink) and from server j to the application (downlink) throughout the process of offloading and retrieving results.

RTT is the period of time required for data to be transmitted from the application to the server and then received back for all jobs. Hence, the calculation of the bandwidth requirement for each task is performed using the following method.

$$P_{ij}\left(RTT + T_i^e\right) \leq dc_i \tag{7}$$

The bandwidth need for a task t_i can be determined using the following method.

$$b_{ij}^w \geq \frac{data_i}{dc_i \frac{wd_i}{R_j^{mec}}} \tag{8}$$

In order to minimise the costs related to each MEC server j, we guarantee the timely completion of all jobs according to their particular timeline limits. Afterwards, The necessary data transfer capacity between the application and the MEC server can be computed.

$$b_{ij}^w = \frac{data_i}{dc_i - \frac{wd_i}{R_j^{mec}}} \tag{9}$$

Because there is a bandwidth limit on each MEC server, the total bandwidth used by all scheduled tasks on a server must not exceed its bandwidth constraint.

$$\sum_{i=1}^{N} p_{ij} B_{ij}^w \leq B_{mic} \tag{10}$$

The Mobile Computing Agent (MCA) acts as a coordinator, establishing communication with each MEC server and regularly monitoring them. Each MEC server's cost is mostly based on two things: how well it functions and the necessary resources for implementing microservices for each task that is assigned to it. The cost for a MEC server is not revealed. merely by its operating status. MCA charges customers only for the specific capabilities they require, rather than the whole cost of the server. The binary variable φ_j is used to indicate the on/off state of MEC server j.

$$\varphi_j = \begin{cases} 1, & m_i \leftarrow On; \\ 0, & Off \end{cases} \tag{11}$$

MEC cost model

Microservices are discrete autonomous software components that are not standalone computational applications. The components encompassed in this are the cost model, which elucidates the method of accessing resources on-demand, guaranteeing connectivity according to the framework with regard to commercial applications. Equation 12 depicts the cost paradigm for mobile applications based on demand. Specifically focusing on business applications. The model performed calculations to determine the processing requirements for each application that was chosen for the experiments.

$$C_j = \varrho_j * p_{ij} * T_i^e \tag{12}$$

The ϱ_j values in Table 3 represent the cost per unit of computational work for each unique MEC server.The resource restrictions for each MEC server must now be calculated, eqs. 13 to 25 are used to determine the computed constraints. The resource limitations for MEC servers MEC_1 to MEC_3 are calculated using the formulae below. Distribution of resource allocation is necessary to manage expenses, bandwidth needs, and processing duties based on microservices across all resources. The decision regarding task offloading is made by the P_{ij}= 0,1 based on the specific criteria listed in Table 2. Equation 13 calculates minimum resource capacity Z required for each task based on the resources demand. The minimum resource consumption Z is dependent on the On-Demand ϱ_j and the state of the MEC Server ϱ_j. Equation 14 calculates the minimum value of Z required for the task at hand. Equ 16 calculates computational time required for each task on a given server. Equation 17 is utilized to calculate the performance of the i^{th} task in mobile applications based on the data provided in table 3. Equations 18 and 19 calculate The process of comparing and executing tasks on MEC resources, as well as the resources needed to compete against each task. Furthermore, eqs. 20 and 21 calculate the j^{th} MEC server's capacity for resources and set it to a value of 1. This indicates that the resource are determine transferred to server of MEC, eqs. 22, 23, and 24 determine The required server, bandwidth, and virtual machine (VM) resources for each task based on microservices architecture to be transferred to the cloud. The ultimate determination calculated $p_{ij} = 0, 1$ in order to obtain offloading task with necessary resources.

$$min\ \ Z = \sum_{a=1}^{M} \sum_{i=1}^{N} \varphi_j.C_j \forall \imath \in N \tag{13}$$

$$Subjecttomin\ \ Z = \sum_{a=1}^{M} \sum_{i=1}^{N} \varphi_j P_{ij}.C_j \forall \imath \in N \tag{14}$$

$$T_{j,0} = 0, \quad \forall \{j = 1, \ldots, N\} \tag{15}$$

$$T_j^k = T_j^k - 1 + \sum_{k=1}^{N} p_j^k T_k^e \quad \forall \{i = 1, \ldots, m\} \tag{16}$$

Table 3 MEC servers unit price [42]

MEC server	On-demand ϱ_j			State
	CostPerBw	StorageCost	CPUCost	φ_j
m_1	0.4	0.2	2	1/0
m_2	0.5	0.5	4	1/0
m_3	0.7	0.6	5	1/0

$$T^{e}_{t_i} = \sum_{j=1}^{M} p_{ij} * \frac{W_{di}}{\varsigma j} \quad \forall\{i = 1, \ldots, n\} \tag{17}$$

$$M_{E_i} = \sum_{i=1}^{M} T^{j}_{k} p_{ij} \quad \forall\{i = 1, \ldots, N\} \tag{18}$$

$$M_{E_i} + RTT \le dc_i \tag{19}$$

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

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

$$\sum_{j=1}^{N} p_{ij} sd_i \le S_{mj} = 1 \quad \forall i \in 1, 2, \ldots M \tag{22}$$

$$\sum_{i=1}^{N} p_{ij} \le V_{j_{mic}} = 1 \quad \forall i \in 1, 2, \ldots M \tag{23}$$

$$\sum_{j=1}^{N} p_{ij} b^{w_j}_{mec} \le B_{mic} \quad \forall i \in 1, 2, \ldots M \tag{24}$$

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

Proposed OTSCOCA framework and system

The OTSCOCA framework for Optimisation Task Scheduling and Computational Offloading with Cost Awareness includes several components, as depicted in Fig. 2. In the first step, mec server resources are matched with each task using a pair-wise approach [43]. The task sequence module plays a vital role in organising tasks in a specific order, ensuring optimal scheduling by the scheduler. Task t_i will be scheduled on server j if the condition P_{ij} is met, otherwise it will not be scheduled. This process continues until all requested tasks are completed within their specified deadlines in the MCBMEC. The MEC servers utilise various components to efficiently process applications and optimise scheduling. These components are outlined in Algorithm 1.

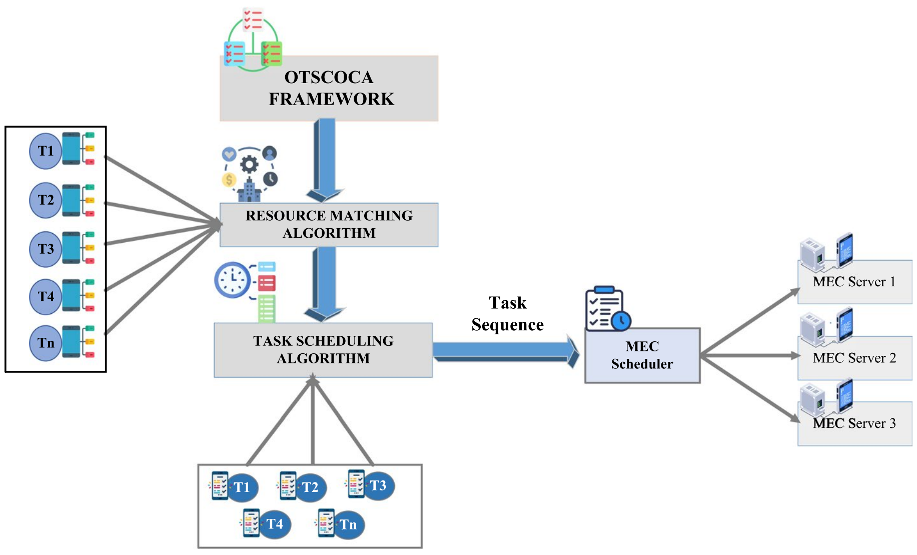

Fig. 2 OTSCOCA framework

Algorithm 1 OTSCOCA framework

```
Input:
for each application $G$ in set $A$, for each task $T$ in $G$, for each MEC server $J$
in set$M$
1  Output: { min Z}
2  begin
3      $\sum_{i=1}^{M}\sum_{j=1}^{N} P_{ij} * \xi_j * C_j$;
4      Task Sequencing $(M, T, G)$;
5      Task Scheduling $(G, T, M)$;
6      $PLIST \leftarrow null$;
7      foreach $(M)$ do
8          foreach $(G)$ do
9              foreach $(T)$ do
10                 if $(T \neq \omega)$ then
11                     then here we call the ;
12                     Resource Matching Algorithm;
13                     $PLIST[] \leftarrow t_i, \leftarrow t_j$
14                 if $(TPLIST[] \neq \omega)$ then
15                     then here we call the ;
16                     Task Sequence Algorithm;
17             Call Task Scheduler;
18             Compute $Z^*$ using the values stored in the$\leftarrow PLIST[]$ according to
                 the equation (13);
19         $Z \leftarrow Z^*$;
20     return Z;
21 Endloop;
```

MEC server resource attaining

Our focus lies on addressing the cost optimization problem in the context of heterogeneous MEC servers. To achieve the best results, it is advisable to use the most effective MEC server for carrying out all the long sequential operations. The primary objective of our microservices application task scheduling is to reduce the computational and processing expenses associated with MEC. The process of selecting the MEC server with the minimum cost ϱ_j is a complex task. Equation 26 represents the unit cost ϱ_j while eq. 27 depicts the smaller MEC server costs.

$$\varrho_j = \frac{C_j}{\sigma_j} \tag{26}$$

In eq. 26 σ_j represents the size of the MEC server M_j. The server's cost is determined by factors such as its processing power, memory capacity and size of tasks so it can handle. The result of evaluating eq. 26 yields the aggregate cost incurred by the chosen MEC server. The unit cost ϱ_j is calculated using eq. 27. The unit cost is calculated based on the demand for MEC tasks the demand for MEC virtual machines and the demand for MEC bandwidth handling. This concept was developed to facilitate the delegation of tasks based on their computed cost.

$$\sigma_j = \frac{MEC_{Sm_j}}{\sum_{i=1}^{M} s_{mj}} + \frac{MEC_{V_{jmic}}}{\sum_{i=1}^{M} Vj_{mic}} + \frac{MEC_{B_{mec}}^{wj}}{\sum_{i=1}^{M} B_{mec}^{wj}} \tag{27}$$

Upon the completion of successful processing on the servers any surplus resources are no longer required and should not be squandered. h_{mj} refers to the remaining resources available on the MEC server M_i.Once the assignment scheduling for the initial level is finalized one objective of a given task is to optimize the dot product ψ_i [44] which serves as a representation of the products and their fundamental operations. In order to determine the values of ψ_i, eqs. 11, 12, 13 and 14 are utilized in the subsequent equations exhaustive declarations for the proposed system are illustrated.

$$\psi = \overline{h_{mj}}\gamma\overline{t_{ij}} \tag{28}$$

$$\psi = s_{di}S_{mj}^{\wr}q + v_j^{\wr}q + b_{ij}^{w}B_j^{\wr}q \tag{29}$$

$$h_{m_j} = (S_{mj}^{\wr}q, S_j^{\wr}q, B_j^{\wr}q) \tag{30}$$

$$h_{m_j} = \gamma_{m_j} - \sum \gamma t{*}j \tag{31}$$

The variable $\gamma t{*}j$ represents the allocation of tasks for resource management and scheduling on the MEC server M_j. The MEC server system encompasses a variety of types and resources. Resource matching is a process that is employed in the MEC server to assign the most suitable and highest quality resource for every task taking into consideration the diverse and varied characteristics of the resources available. The tasks consist of vector attributes including task deadline, data size and workload. In contrast the resources encompass vector attributes including bandwidth, cost, virtual machine capacity and storage. Resource matching is a critical issue that requires attention and resolution. The resource matching algorithms chosen for this task are the Techniques to execute the multi-criteria decision of resource matching. We utilize Analytic Hierarchy Processing (AHP) [43] and the (TOSS) technique [45]. Algorithm 2 is specifically developed for the purpose of matching MEC server resources. The system accepts inputs in the form of resources and tasks which are ordered sequentially. The result produced by the algorithm is an array called *PList*[] which represents the frequent list. The *PList*$[t_i, k_j]$ array is populated with the necessary data after the completion of the task's requirements. This data is then stored on the MEC server.

Algorithm 2 includes a step in which the requirements of MEC server are validated for each incoming task t_i. Assuming the MEC server determines a good match between the resources algorithm will return a boolean value of true, otherwise it will return false. After the successful retrieval of accurate results, we add the corresponding list to the frequent list *PList* $[k_j, t_i]$. Step 4 involves iterating through all potential tasks originating from mobile devices in order to determine if they align with the diverse requirements of the MEC server.

Algorithm 2 Matching resources of MEC servers

```
  Input: Consider Collection applications $G$ from category $A$,;
  tasks denoted by $T$ within each application subset $G$;
  $m_j \leftarrow \{Sd_i, wd_i, dc_i\}$
  $m_j \leftarrow \{Sm_j, V^j_{mec}, B^w_j\}$
  Output: PLIST[]
1 begin
2     foreach (A) do
3         foreach (G) do
4             if $(t_i == m_j = true)$ then
5                 Applied the analytic hierarchy process
                  Applied the echnique for order performance
                  by similarity to ideal solution
                  Add $PLIST[t_i, m_j]$
                  else
6                     Repeat step-4 again
7             $PLIST[t_i, k_j]$;
8         Endloop;
```

MEC server task sequencing

Tasks are entered into the system in a random manner, following a Poisson process. Offloaded tasks are submitted to the system without any specific order. As a result, it becomes necessary to implement a task sequencing process. Our task sequencing method is based on three rules that take into account important factors such as the deadline, size, and slack time. Afterwards, we establish three rules to categorize the submitted tasks according to these characteristics, as outlined below:

- Earliest Due Date(EDD): We prioritise the tasks in the set based on their deadlines, giving higher importance to tasks with smaller deadlines. When deadlines are the same, tasks with lower sizes are given greater priority. We evaluate the tardiness of tasks using the following methodology.
- Lateness time first (LTF): Tasks are organised according to their lateness time, giving priority to those with the shortest lateness time for early scheduling.
- The Shortest Size First (SSF):The order in which tasks are scheduled is based on their size, with smaller tasks being prioritised and scheduled before larger ones.

Algorithm 3 Optimising task scheduling for cost-efficient

```
   Input: Applications: $G \in A$, Tasks $T$, MEC Servers $M$
   Output: $P_{ij}$ Task Scheduling, $y_j$ Condition of MEC server
1  begin
2      Declare each and every binary variable $P_{ij}$ to 0;
3      Preliminary Solution $Z$;
4      MEC server state $y_j$ to 0;
5      Utilize the vector attribute $p_{ij}$;
6      Calculate the unit cost $mu_j$ of MEC server's belonging to the set $M$;
7      Calculate of MEC servers that being utilization for exploitation $F = \{\}$;
8      foreach $(A)$ do
9          foreach $(T)$ do
10             foreach $(M)$ do
11                 if $(T \in G \neq \phi)$ then
12                     Select one with lowest cost $\varrho_j$;
13                     $t_i \leftarrow$m$_j$;
14                     Calculate the value of $\psi$ for each task to $m_j$;
15                     Assign task $t_i$ to MEC server $m_j$ based on the highest dot
                        product size;
16                     $T \leftarrow T$;
17                     Establish $p_{ij=1}$;
18                     Establish $x_{j=1}$;
19                     $F = F \cup \{m_j\}$;
20                     $Z \leftarrow p_{ij}$ efficient assignment;
21                 $M = M \cup \{m_j\}$;
22             Obtain chosen MEC servers, $m_{g1}$ and $m_{g2}$,at a minimal cost unit
                denoted by $F$;
23             if $(m_{g1} \geq m_{g2} \in F)$ then
24                 Interchange: $t_i \leftarrow m_{g1}$ to $t_i \leftarrow m_{g1}$ ;
25                 Allocate $(p_{i(g_1)} = 1)$;
26                 $Z^* \leftarrow p_{i(g_1)}$ Ideal assignment;
27             else if $(t_i \leftarrow m_{g2=\phi})$ then
28                 Group $F \leftarrow F|m_{mg2}|$;
29                 Group $x_{mg2} = 0$;   Obtain most recent MEC server;
30                 $m_{g2}$ at a minimal expense within $F$;
31             End Application Assignment;
32     return
```

The MEC cloud system receives offloaded tasks in a random order, and the Mobile Computing Agent (MCA) organises them using a first-in-first-out (FIFO) arrangement with task sequencing. The MCA follows a set of sequence rules in a systematic order to ensure efficient and cost-effective execution. Figure 3 provides a comprehensive overview of the entire process, showcasing the different methods used for task sequencing after offloading. Various task sequencing methods result in varying scheduling results. Therefore, our goal is to choose the most efficient task sequence for offloaded tasks that meets all the necessary requirements and achieves the desired outcome.

Task scheduling

Once the task sequencing and resource matching steps are finished, we are left with an initial task scheduling approach. However, this initial scheduling may not be the most cost-effective solution for mobile applications. As a result of changes in network conditions and cloud resources the initial solution becomes unstable. Thus, additional enhancements are required through creative solutions. For example, let's look at two tasks, t_1 and t_2 that are scheduled to run on two different cloud servers k_1 and k_2 which have different capabilities. The resource demand attributes for tasks t_1 and t_2 are as follows: task t_1 has a deadline of 24, a data size of 15mb, and requires 12 units of CPU. On the other hand, task t_2 has a deadline of 50, a data size of 35mb, and requires 32 units of CPU. On the other hand, the resource attributes for servers k_1 and k_2 are (S_{mj}: 15, V_j^j: 4, B_j^w: 10) and (S_{mj}: 20, V_j^j: 8, B_j^w: 20), respectively.

During the initial scheduling phase task t_i is assigned to MEC server k_1 and then task t_2 is allocated to MEC server k_2. As a result the total cost of the application

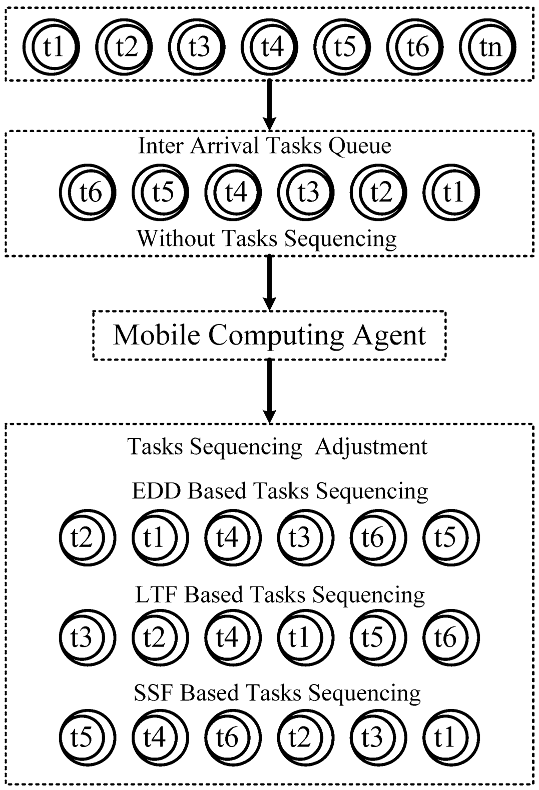
t1
t2
t3
t4
t5
t6
tn
Inter Arrival Tasks Queue
t6
t5
t4
t3
t2
t1
Without Tasks Sequencing
Mobile Computing Agent
Tasks Sequencing Adjustment
EDD Based Tasks Sequencing
t2
t1
t4
t3
t6
t5
LTF Based Tasks Sequencing
t3
t2
t4
t1
t5
t6
SSF Based Tasks Sequencing
t5
t4
t6
t2
t3
t1

Fig. 3 Sequence of the tasks

includes the expenses of both MEC servers. On the other hand, choosing to perform all tasks on MEC server k_2 alone has a significant impact on the overall application cost highlighting the decrease in computation expenses. During the initial scheduling phase, task t_i is assigned to MEC server k_1 and then task t_2 is assigned to MEC server k_2. This leads to the overall application cost which includes the combined expenses related to both MEC servers. Nevertheless, if all tasks are designated to be executed primarily on MEC server k_2 the total cost of the program is determined solely by the cost of MEC server k_2.

The example provided demonstrates that the MEC server selected by the scheduler has low resource execution costs. However, there is a need for further optimization in the initial scheduling to reduce high costs. MEC servers that are initially chosen can often result in significant resource expenses. In order to tackle this issue, we present a task scheduling approach that focuses on improving the efficiency of chosen MEC servers. The main objective is to rearrange the tasks of the application on MEC servers in order to minimize costs starting from the initial scheduling phase. The scheduler algorithm optimizes to utilization the MEC servers with higher costs, thereby minimizing the additional expenses during the initial stage. In order to enhance the cost of the application, we suggest implementing a dedicated task scheduling algorithm meticulously crafted to address our specific optimization challenge. This technique is detailed in Algorithm 2. The algorithm accepts a arranged sequence of tasks in need of scheduling on MEC servers with different capabilities. Here is a detailed breakdown of the execution process for Algorithm 3.

- In lines 1–5, we define several variables. The MEC servers are organized in a descending order following eqs. 27 and 28 which consider ϱ_j and C_j.
- We collect data on every application task, encompassing their specs and resource demands, within lines 7–9.
- The method chooses the lowest-cost MEC server ϱ_j if there are unscheduled jobs in the set T of application G. The selection of the MEC server for executing task scheduling for unscheduled tasks relies on the specific resource needs and the currently available resources of the MEC server. Afterwards, the algorithm chooses the server m_j selecting the MEC server with the most economical unit cost from the pool of accessible servers in set M. In case the resources of selected MEC server are sufficient to handle the resource requirements of certain tasks in the unscheduled tasks set T then it gives priority to the most important task for insertion into server m_j. Furthermore, the server m_j is excluded from the set of possible servers M, as defined by lines 10–19.
- Finding the two MEC servers with the lowest costs in E, the algorithm identifies the last one, m_{g1} and the other m_{g2}. If the resources on server m_{g1} are sufficient to complete task t_i on MEC server m_{g2} the algorithm will transfer task t_i to server m_{g1} and modify the job scheduling variable accordingly. The MEC server m_{g2} is removed from E and its state is updated when all of its jobs are removed. Lines 20–27 specify E and from there we get a new server with the lowest cost.

OTSCOCA time complexity and overhead

The OTSCOCA (The optimization Task Scheduling and Computational Offloading with Cost Awareness) comprises various modules, including Resource Matching, Task Sequences, and Task Scheduling. To find the time complexity, we first look at all three parts separately, and then we combine them. To begin with, resource matching is expected to be carried out by heterogeneous servers using TOPSIS and AHP approaches. The Resource Matching's time complexity algorithm is determined to be $0(TxM)$. M represents the server resources of the Multi-Criteria Decision Making MEC system while T denotes tasks organized an a Matching in pairs. In task sequencing tasks are arranged in ascending order based on their deadlines, size and lateness using the $O(mlogn)$ algorithm. Here n represents total count of the tasks that have been sorted while M refers to the specific method that has been utilized to sort the tasks sequentially order. Task Scheduling: The scheduling of all the MEC servers is based on the order arranged in decreasing sequence ϱ_j and C_j. The computational complexity assessed $O(logM) : O(logM) + N$. The computational complexity refers to the computational efficiency of scheduling tasks on all MEC servers based on the arrangement is determined by prioritizing cost and load reduction. Variable n represents number of tasks in the swapping process which affects computational complexity of various MEC servers.

Performance evaluation

To assess the efficacy of the recommended OTSCOCA and MCBMECS system, functional outcomes were obtained by conducting experiments through various performance evaluations on diverse benchmarks of mobile applications within the system. The experimental setting in this study is divided into separate sections: (i) MCBMECS deployment (ii) Calibration of Components and Metric Parameters (iii) Computational Offloading Framework evaluation, and (iv) Task Scheduling and Algorithmic Comparison. Table 4 contains simulation settings, Table 5 presents the resource specifications of MEC servers obtained from [42], and Table 6 provides a detailed summary of the workloads generated for healthcare, augmented reality (i.e., face recognition), 3D games (e.g., Sudoku, Queen), and e-commerce applications used in this work.

Table 4 Simulation parameters

Simulation parameter	Values
Windows operating system	Windows docker engine
Simulation time	16 h
Experiment repetition	140 times
Android phone	Redmi Note 11 Pro
Processor	X64 bit
Languages	Java, XML, JSON
Duration of simulation	90 h
Monitoring of simulation	Every 2 h
Application interface	One UI 6.0
Method evaluation	ANOVA single and multi-factor
Android OS	Upside Down Cake
On-Demand	EC2

Table 5 Specifications of MEC servers

Cloud	Core	MIPS/CORE	VMs	Storage(GB)	Cost
j_1	i3	10000	2	1200	0.5
j_2	i5	2000	2	2000	0.8
j_3	i7	3000	2	5000	0.9
j_4	i9	3000	2	10000	0.05

Table 6 Analyzing MEC application workloads

Workload	No: of tasks	C.Ins.(MI)	Data size
Healthcare	900	5.9	600
Augmented reality	730	7.2	700
E-Transport	660	7.9	1100
3D/2D-Game	900	5.3	680

Baseline offloading system and algorithm approaches

In order to conduct a comparison with the current approach, the following primary factors are taken into account when evaluating the obtained results:

- We construct a framework for computational offloading using virtual machines for experimental purposes, making use of the existing frameworks [27]. The goal is to transfer the entire mobile application to the cloud server.
- A framework for dynamic computational outsourcing is implemented using virtual machines, drawing inspiration from the strategies discussed in references [4]. The objective is to transfer whole programs to servers with different characteristics when sufficient resources are available to meet the demands.

We will now compare the various pre-existing task scheduling methods:

- We utilize well-established and economical fixed task scheduling scheduling methods [27] throughout the experimental phase, assessing their performance in relation to the proposed scheme in terms of application costs.
- During the experimental phase, we apply well-established and cost-effective dynamic task scheduling approaches in refs. [29, 30]to evaluate their performance in relation to the proposed scheme, specifically in terms of application costs.
- We use well-proven, Efficient, and affordable static task scheduling techniques [28], omitting task scheduling during the experimental stage, and evaluate their effectiveness in comparison to the suggested plan with regard to application expenses.

Performance metrics

The paper's component calibration involves proposing an experimental approach that randomly generates a diverse set of application tasks, as shown in Table 3. The experiment involves testing with four different numbers of applications. To impose deadline constraints for various task forms, the paper establishes task deadlines using the following equation.

$$d_t^{a,i} = R_{a,i} + \gamma + R_{a,i} \tag{32}$$

The earliest completing duration and a particular percentage of the early completion time are used to calculate the task's deadline $d_t^{a,i}$. A parameter called is used to control the level of strictness for the task deadline. It can be set to values between 0.2 and 1, specifically {0.2, 0.4, 0.6, 0.8, or 1}. As a result, every task is assigned one of five different deadlines, known as $D1$, $D2$, $D3$, $D4$, and $D5$. In order to evaluate the algorithm's performance under various task deadlines, the paper calculates using the formula (31). The paper utilizes RPD (Relative Percentage Deviation) statistical analysis to assess the performance of the *NOMA*, *DCC* and *CTOS* algorithms across different task deadlines. This analysis evaluates the fluctuations in power consumption across various parameters, frameworks and algorithm combinations during the component calibration parameterization. The RPD estimation is shown in Eq. (32)

$$RPD\% = \frac{R_a^* + R_a}{R_a^*} \times 100\%. \tag{33}$$

Here, R_a represents the optimization objective function.

MCBMECS implementation

A mobile cloud-based application is developed and deployed on Mobile Devices using the mobile application developer IDE, specifically android studio. The xiaomi node 12 2023 mobile model is utilized as an emulator for testing purposes. Figure 4 depicts the aforementioned description pertaining to various applications and scenarios. The Edge X Foundry is assessed using an open-source framework. The implementation of the Microservice Container-Based Mobile Edge Computing System MCBMECS framework is comprised of three primary portions. The Mobile Users layer refers to the segment of users who access and utilize mobile devices for various purposes. The Mobile Cloud Agent Control layer pertains to the component responsible for managing and controlling the interactions between mobile devices and the cloud infrastructure. Lastly, the mobile

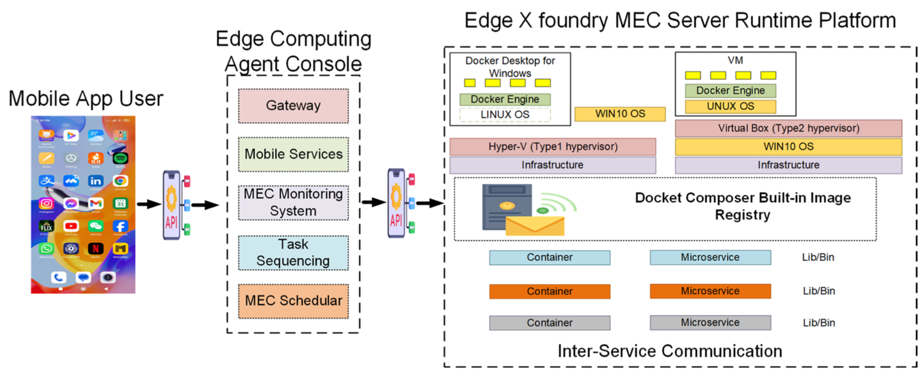

Fig. 4 Implementation OF MCBMECS

cloud resources layer denotes the infrastructure and resources available in the cloud that are utilized by mobile devices for storage, processing, and other computing tasks.

Through the REST API, the mobile apps assign relating tasks to the MCA Console. When communicating with the MCA through a gateway interface, requests and responses are parsed using the JSON format. The API request is parsed in real-time by the console interface. The Mobile Computing Agent is informed by the device services about the precise services needed to finish the offloaded task based on task's attribute. It is the job of the mobile cloud monitoring system to keep an eye on the system's stability and run frequent checks on the tasks list. Task sequencing is the process of arranging tasks in a logical order, while task scheduler is responsible for scheduling tasks to be executed on heterogeneous mobile cloud servers. The run time refers to the operational environment of a system that is based on the specific system scenario. The JVM is responsible for executing Java programs with optimal efficiency. The JVM can be likened to a virtual machine similar to running in a windows docker session. Using these containers, autonomous microservices can be instantiated. Through registry services, the containers are registered with a server, the mobile edge cloud in order to optimize the consumption of all available services. The utilization of REST API facilitates inter-service communication between microservices, resulting in reduced overhead.

Computational offloading framework comparison

The new microservices container-based mobile edge computing for compute offloading demonstrates a decreased startup time in comparison to established frameworks that rely on heavyweight virtual machines. Simultaneously, we have improved the efficiency of resource usage in the suggested system. These elements are supported by both experimental and simulation contexts. Figures 5, 6, and 7 demonstrate the enhancements in boot uptime, overhead, and resource utilization that have been accomplished by our compute offloading approach. The primary determinant influencing these outcomes is the inherent lightweight characteristic of containers as opposed to the resource-intensive virtual machines while executing mobile apps during scheduling. Hence, our

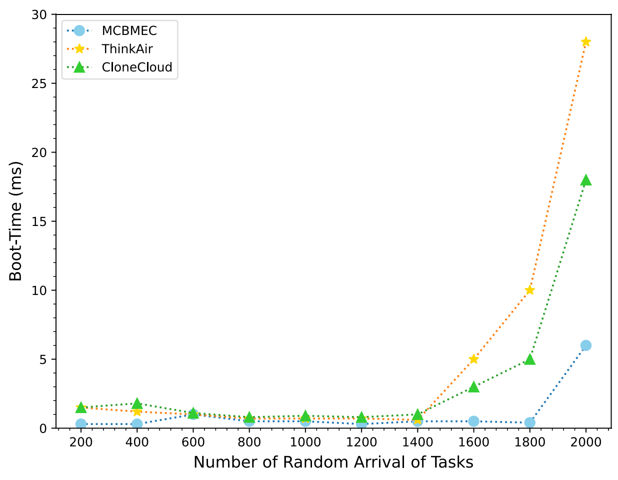

Fig. 5 Boot-Time

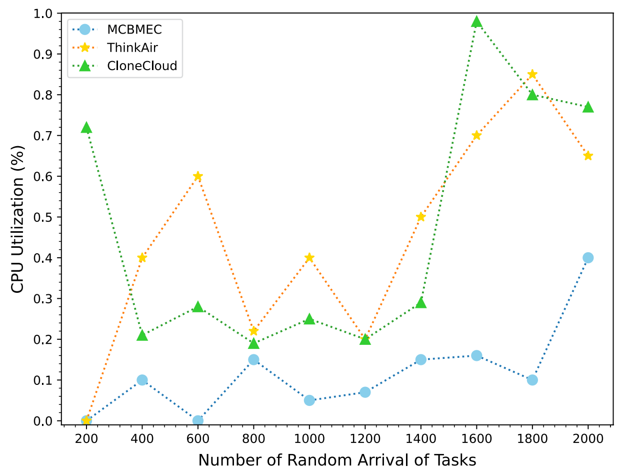

Fig. 6 CPU utilization

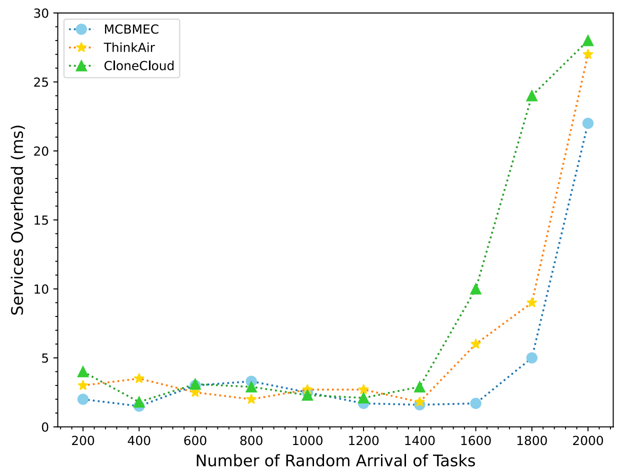

Fig. 7 Overhead

computational offloading architecture demonstrates itself as a proficient platform for executing time-critical applications.

Task sequencing

The rules for sequencing tasks specifically EDD, SPF and SSTF have been optimised in our study to arrange tasks for the purpose of scheduling. In Fig. 8a, the plot of the devised task sequence protocols along with 95.0% Tukey HSD intervals, indicates that the relative percent difference RPD significance of the EDD algorithm is significantly lower compared to the SPF and SSTF algorithms, as depicted in Fig. 8b. This discovery indicates that tasks that are scheduled using the task sequence generated by the EDD rule experience reduced delays in MEC clouds within a heterogeneous environment. As a result, we have chosen EDD as the Mob-Cloud job sequencing component. Prioritisation of tasks is established in our study using the EDD sequence approach. This method assigns higher priority to tasks with the shortest deadline compared to other tasks.

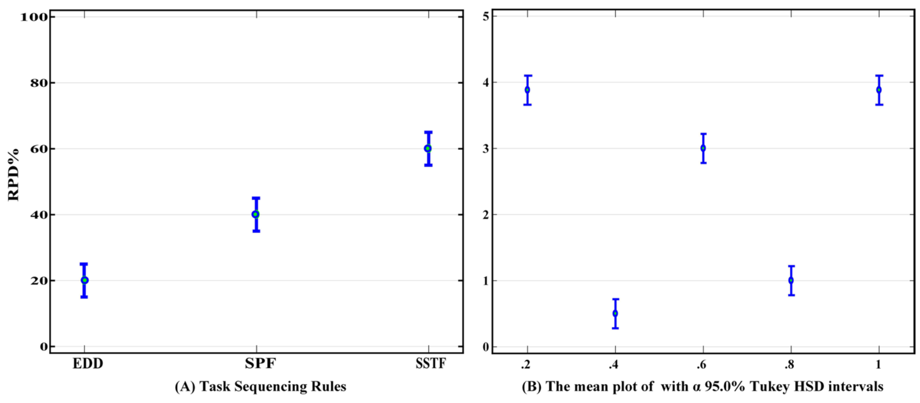

Fig. 8 With the average plot for tasks having 95.0% Tukey HSD intervals, the average graph for application tasks that arrive at random is displayed. Containers and the virtual machine system are contrasted

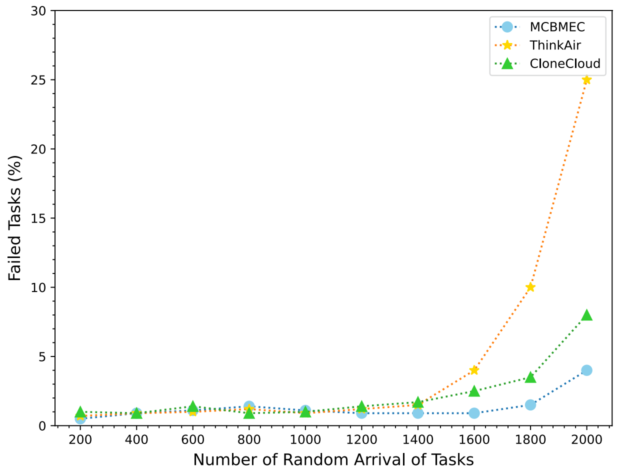

Fig. 9 Failed task during scheduling procedure

Task scheduling

Optimizing task scheduling for cost-effectiveness in mobile applications. The scheduling process takes into account the costs associated with applications, such as communication and computation costs, as well as task deadlines, which are the main concerns. We evaluate various MEC servers for the job scheduling problem. Our objective is to reduce application expenses and ensure their timely completion.

Figures 9 and 10 demonstrate that the proposed OTSCOCA system results in reduced application costs for all applications and ensures that they are executed within their specified timeframes. We take into account the various time limits specified in eq. (31). The primary rationale behind this approach is that the proposed strategy systematically enhances Improve solutions iteratively until you get the best result. We enhance the task failure ratio in our task scheduling method in comparison to the previous research. When an existing baseline is nearing, simply take into account

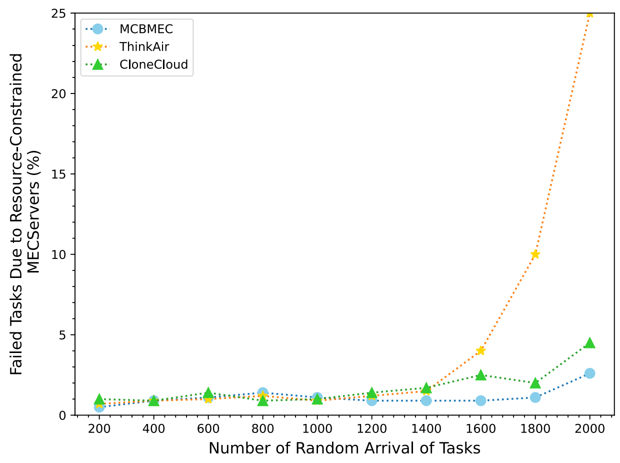

Fig. 10 Task failed due to insufficient resources

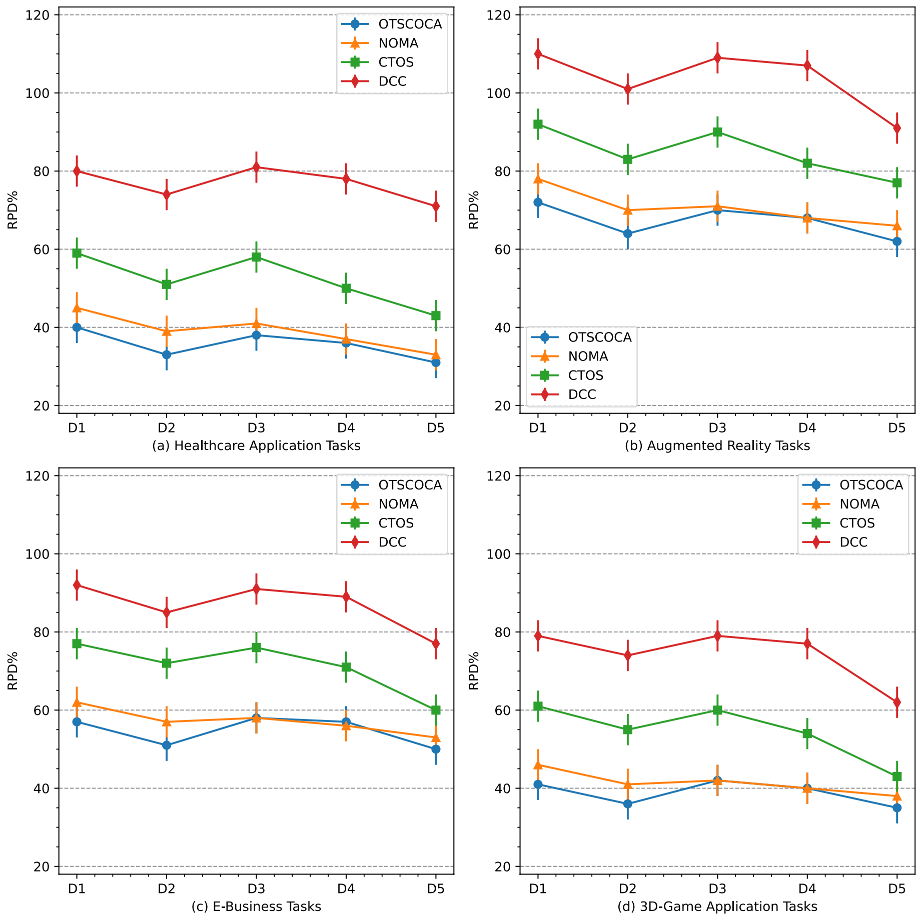

Fig. 11 Varying deadlines for tasks across different applications

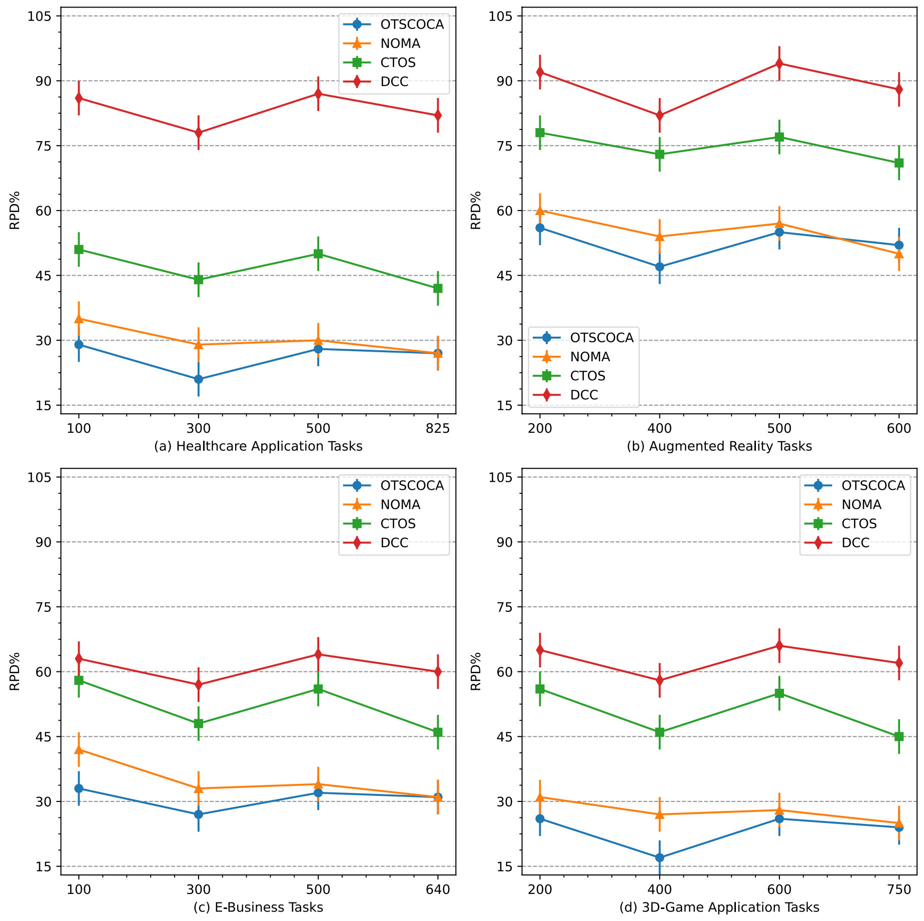

Fig. 12 Obtaining the overall cost of all tasks

the original solution. Figures 11 and 12 illustrate that the proposed system exhibits a lower failure ratio for tasks after the experiment, in comparison to the baseline technique. Therefore, the suggested dynamic OTSCOCA approach is effective in a dynamic scenario and enhances cost efficiency while meeting deadlines.

Conclusion

The use of mobile applications has seen a notable rise, catering to a range of purposes including E-banking, Augmented Reality, E-Transportation, and E-Healthcare. Conventional cloud-based frameworks that depend on Virtual Machines often encounter issues such as long boot times, excessive overhead, and unnecessary costs associated with running mobile applications. To tackle these concerns, the study suggests the implementation of Microservices Container-Based Mobile Edge Computing Applications MCBMEC. Our primary objective is to address the task scheduling challenges that arise when dealing with a variety of Mobile Edge Servers. In addition, the study presents the OTSCOCA framework, which enhances task scheduling and computational offloading by considering cost factors through sequential processes such as Resource Matching, Task Sequencing, and Task Scheduling. The results obtained from the experiments highlight the success of MCBMEC and OTSCOCA, illustrating improved server utilization, decreased service latency, and more efficient average service bootup times. These enhancements make a substantial impact on reducing expenses. Furthermore, the experiments demonstrate the significant improvements achieved by the MCBMEC and OTSCOCA methods in terms of server efficiency, service latency reduction, average bootup time improvement, and cost optimization.

In future, it is crucial to take into account the security and privacy considerations associated with mobile edge computing applications in order to establish reliable and trustworthy systems in the future.

Acknowledgements
This work is supported by the National Key Research and Development Program (No. 2022YFB3305500), the Key-Area Research and Development Program of Guangdong Province (No. 2021B0101200003), the National Natural Science Foundation of China (Nos. 62273089, 62102080), Natural Science Foundation of Jiangsu Province (No. BK20210204)

Author contributions
AS developed the research idea, implemented the algorithm, designed and executed the experiments, and drafted the manuscript. DKS helped with the data collection and labeled the dataset provided fruitful comments on the interpretability of the schematic fgures. LX provided general research guidance, managed the research workfow, and revised the initial versions of the manuscript. All authors read and approved the final manuscript.

Funding
No funding.

Data availability
No datasets were generated or analysed during the current study.

Declarations

Competing interests
The authors declare no competing interests.

Received: 21 January 2024 Accepted: 24 July 2024
Published online: 04 September 2024

References
1. Sufyan F, Chishti MS, Banerjee A. Energy and delay aware computation offloading scheme in mcc environment. 2022. p. 247–54.
2. Liu J, Ren J, Zhang Y, Peng X, Zhang Y, Yang Y. Efficient dependent task offloading for multiple applications in MEC-cloud system. IEEE Trans Mob Comput. 2021;22(4):2147–62.
3. Badshah A, Iwendi C, Jalal A, Hasan SSU, Said G, Band SS, Chang A. Use of regional computing to minimize the social big data effects. Computers Ind Eng. 2022;171:108433.
4. Huang X, Lei B, Ji G, Zhang B. Energy criticality avoidance-based delay minimization ant colony algorithm for task assignment in mobile-server-assisted mobile edge computing. Sensors. 2023;23(13):6041.
5. Gong Y, Bian K, Hao F, Sun Y, Wu Y. Dependent tasks offloading in mobile edge computing: a multi-objective evolutionary optimization strategy. Future Gener Computer Syst. 2023;148:314–25.
6. Yang C, Chen Q, Zhu Z, Huang Z-A, Lan S, Zhu L. Evolutionary multitasking for costly task offloading in mobile-edge computing networks. IEEE Trans Evol Comput. 2023;28(2):338–52.
7. Sajnani DK, Mahesar AR, Lakhan A, Jamali IA, Lodhi R, Aamir M. Latency aware optimal workload assignment in mobile edge cloud offloading network. 2018. p. 658–62.
8. Almanifi ORA, Chow C-O, Tham M-L, Chuah JH, Kanesan J. Communication and computation efficiency in federated learning: a survey. Internet Things. 2023;22:100742.
9. Salami Y, Khajehvand V, Zeinali E. E3c: a tool for evaluating communication and computation costs in authentication and key exchange protocol. Iran J Computer Sci. 2024. https://doi.org/10.1007/s42044-024-00176-x.
10. Shao J, Zhang J. Communication-computation trade-off in resource-constrained edge inference. IEEE Commun Mag. 2020;58(12):20–6.
11. Akhlaqi MY, Hanapi ZBM. Task offloading paradigm in mobile edge computing-current issues, adopted approaches, and future directions. J Netw Computer Appl. 2023;212:103568.
12. Feng C, Han P, Zhang X, Yang B, Liu Y, Guo L. Computation offloading in mobile edge computing networks: a survey. J Netw Computer Appl. 2022;202:103366.
13. Wang J, Pan J, Esposito F, Calyam P, Yang Z, Mohapatra P. Edge cloud offloading algorithms: issues, methods, and perspectives. ACM Computing Surv (CSUR). 2019;52(1):1–23.
14. Rezaee MR, Hamid NAWA, Hussin M, Zukarnain ZA. Fog offloading and task management in IOT-Fog-cloud environment: review of algorithms, networks and SDN application. IEEE Access. 2024. https://doi.org/10.1109/ACCESS.2024.3375368.

15. Yin L, Sun J, Wu Z. An evolutionary computation framework for task off-and-downloading scheduling in mobile edge computing. IEEE Internet Things J. 2024. https://doi.org/10.1109/JIOT.2024.3381187.
16. Prajapati A, Patel DM. Container scheduling: a taxonomy, open issues and future directions for scheduling of containerized microservices. Open Issues and Future Directions for Scheduling of Containerized Microservices. 2024.
17. Cui H, Tang Z, Lou J, Jia W, Zhao W. Latency-aware container scheduling in edge cluster upgrades: a deep reinforcement learning approach. IEEE Trans Serv Comput. 2024. https://doi.org/10.1109/TSC.2024.3394689.
18. Savusalo T. Application for managing container-based software development environments. Master's thesis. 2023.
19. Urblik L, Kajati E, Papcun P, Zolotová I. Containerization in edge intelligence: a review. Electronics. 2024;13(7):1335.
20. Bentaleb O, Belloum AS, Sebaa A, El-Maouhab A. Containerization technologies: taxonomies, applications and challenges. J Supercomput. 2022;78(1):1144–81.
21. Vhatkar KN, Bhole GP. Optimal container resource allocation in cloud architecture: a new hybrid model. J King Saud Univ-Computer Inf Sci. 2022;34(5):1906–18.
22. Sufyan F, Banerjee A. Computation offloading for distributed mobile edge computing network: a multiobjective approach. IEEE Access. 2020;8:149915–30.
23. Alkhalaileh M, Calheiros RN, Nguyen QV, Javadi B. Data-intensive application scheduling on mobile edge cloud computing. J Netw Computer Appl. 2020;167: 102735.
24. Sufyan F, Banerjee A. Computation offloading for smart devices in fog-cloud queuing system. IETE J Res. 2023;69(3):1509–21.
25. Yang G, Hou L, He X, He D, Chan S, Guizani M. Offloading time optimization via Markov decision process in mobile-edge computing. IEEE Internet Things J. 2020;8(4):2483–93.
26. Liu J, Li C, Luo Y. Efficient resource allocation for IoT applications in mobile edge computing via dynamic request scheduling optimization. Expert Syst Appl. 2024;255:124716.
27. Lou J, Tang Z, Zhang S, Jia W, Zhao W, Li J. Cost-effective scheduling for dependent tasks with tight deadline constraints in mobile edge computing. IEEE Trans Mob Comput. 2022;22(10):5829–45.
28. Zhang E, Zhao L, Lin N, Zhang W, Hawbani A, Min G. Cooperative task offloading in cybertwin-assisted vehicular edge computing. In: 2022 IEEE 20th International Conference on Embedded and Ubiquitous Computing (EUC). IEEE; 2022. p. 66–73.
29. Naouri A, Wu H, Nouri NA, Dhelim S, Ning H. A novel framework for mobile-edge computing by optimizing task offloading. IEEE Internet Things J. 2021;8(16):13065–76.
30. Qian LP, Shi B, Wu Y, Sun B, Tsang DH. Noma-enabled mobile edge computing for internet of things via joint communication and computation resource allocations. IEEE Internet Things J. 2019;7(1):718–33.
31. Badshah A, Jalal A, Farooq U, Rehman G-U, Band SS, Iwendi C. Service level agreement monitoring as a service: an independent monitoring service for service level agreements in clouds. Big Data. 2023;11(5):339–54.
32. Chun B-G, Ihm S, Maniatis P, Naik M, Patti A. Clonecloud: elastic execution between mobile device and cloud. In: Proceedings of the sixth conference on Computer systems. 2011. p. 301–14.
33. Kosta S, Aucinas A, Hui P, Mortier R, Zhang X. Thinkair: dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In: 2012 Proceedings IEEE Infocom. IEEE; 2012. p. 945–53.
34. Lee H-S, Lee J-W. Task offloading in heterogeneous mobile cloud computing: modeling, analysis, and cloudlet deployment. IEEE Access. 2018;6:14908–25.
35. Stavrinides GL, Karatza HD. An energy-efficient, QoS-aware and cost-effective scheduling approach for real-time workflow applications in cloud computing systems utilizing dvfs and approximate computations. Future Gener Computer Syst. 2019;96:216–26.
36. Sathya Sofia A, GaneshKumar P. Multi-objective task scheduling to minimize energy consumption and makespan of cloud computing using NSGA-II. J Netw Syst Manag. 2018;26:463–85.
37. Guo M, Mukherjee M, Lloret J, Li L, Guan Q, Ji F. Joint computation offloading and parallel scheduling to maximize delay-guarantee in cooperative MEC systems. Digit Commun Netw. 2022. https://doi.org/10.1016/j.dcan.2022.09.020.
38. Porambage P, Okwuibe J, Liyanage M, Ylianttila M, Taleb T. Survey on multi-access edge computing for internet of things realization. IEEE Commun Surv Tutor. 2018;20(4):2961–91.
39. Peng H, Wen W-S, Tseng M-L, Li L-L. Joint optimization method for task scheduling time and energy consumption in mobile cloud computing environment. Appl Soft Comput. 2019;80:534–45.
40. Sahito MA, Kehar A. Dynamic content enabled microservice for business applications in distributed cloudlet cloud network. Int J. 2021;9(7):1035–9.
41. Patsias V, Amanatidis P, Karampatzakis D, Lagkas T, Michalakopoulou K, Nikitas A. Task allocation methods and optimization techniques in edge computing: a systematic review of the literature. Future Internet. 2023;15(8):254.
42. Amazon elastic compute cloud (amazon ec2). 2023. https://aws.amazon.com/ec2/pricing/on-demand/. Accessed 3 Jan 2024.
43. Saaty TL. Decision making with the analytic hierarchy process. Int J Serv Sci. 2008;1(1):83–98.
44. Pearce DJ, Kelly PH. A dynamic topological sort algorithm for directed acyclic graphs. J Exp Algorithm. 2007;11:1–7.
45. Yonghui Q. A study for the multi-attribute decision-making method based on TOPSIS. Technol Dev Enterp. 2006;25(9):89–91.

Publisher's Note