SN Computer Science (2024) 5:330
https://doi.org/10.1007/s42979-024-02600-2

ORIGINAL RESEARCH

Framework for Agent-Based Multistage Application Partitioning Algorithm in Mobile Cloud Computing

Asia Kanwal[1] · **Tehmina Amjad**[1] · **Humaira Ashraf**[1]

Received: 17 October 2023 / Accepted: 30 December 2023 / Published online: 23 March 2024

Abstract
The idea of computational offloading is quickly catching on in the world of mobile cloud computing (MCC). Today's applications have heavy demands on power and computing resources, creating issues with energy consumption, storage capacity, and mobile device performance. Mobile devices may efficiently offload their calculations to cloud servers using the offloading paradigm and then get the processed results back onto the device. The investigation relates to identifying the specific application components that should be offloaded and run remotely and which parts are supposed to be treated locally. The applications need to be partitioned to differentiate between remote and local codes. In this paper, an agent-based multistage graph partitioning (ABMP) scheme is proposed. The framework of the scheme is based on three-tier architecture that includes mobile, cloudlet, and cloud for the execution of application tasks. The main goal is to provide an efficient partitioning and offloading scheme in the mobile cloud computing area. The results show that incorporating both agent-based multistage graph partitioning and offloading algorithms yields superior performance as compared to previous methods in terms of reducing execution costs and conserving battery life for mobile devices.

Keywords Mobile cloud computing · Application partitioning · Edge computing · Ready time partitioning offloading · Agent-based partitioning · Multistage graph partitioning

Introduction

Cloud computing is frequently used to receive, maintain, and control user data on a distant server. Various computationally intensive applications, such as facial recognition, language processing, online gaming, e-health, etc., are in high demand due to the rise in smart phone users [1]. Mobile edge computing (MEC) was developed for data gathering, processing, power, and delivery to the user-end device to calculate this program on-premise. These computer applications have increased battery energy consumption, storage capacity, and application execution time [2]. Mobile cloud computing (MCC) is viewed as a viable solution to these problems [3]. Numerous studies have shown that MEC may enhance resource allocation and task-offloading mechanisms to increase service quality and energy efficiency. For instance, MEC in very complex networks [4] created a unique user-centric mobility management strategy. A multilayer edge computing architecture and ideal offloading method that reduces job length through the use of a heuristic algorithm is suggested in [5]. In [6], authors suggested minimizing computational latency and energy consumption of user workloads through collaboration between edge and clouds for resource allocation in edge computing. According to [7], their approach evaluated whether or not to offload a task to decrease processing time, the device's battery consumption, and operation cost. However, after the devices have offloaded their work, we decide whether they should move to and be divided among MEC to decrease task execution latency and enhance network throughput. As a result, we discuss a technique in the collaborative edge computing

Tehmina Amjad and Humaira Ashraf have contributed equally to this work.

✉ Asia Kanwal
aasia.phdcs105@iiu.edu.pk

Tehmina Amjad
tehminaamjad@iiu.edu.pk

Humaira Ashraf
humaira.ashraf@iiu.edu.pk

[1] Department of Computer Science and Software Engineering, International Islamic University, H10, Islamabad 44000, Pakistan

architecture that enables activities to be performed outside of the MEC offloaded from the vehicle's communication coverage. Therefore, we take into consideration task partitioning, where a task can be split up into a number of smaller tasks and then calculated concurrently by several MEC. Due to the job execution latency being reduced, this might increase service stability. A device can acquire computation results without experiencing a service interruption before leaving MEC coverage by shortening the execution latency. In the process of partitioning or aggregating the computing results, more usage or energy consumption expenses are incurred, but there are additional benefits, such as a decrease in execution time and an increase in system performance [8].

This study suggests adding an agent to the computing task offloading based on the relevant findings. An agent is something that has sensors to monitor its surroundings and effectors for acting on it. Agents are intelligent program blocks that are active at the terminal device, edge, and cloud levels in this article. They possess qualities including active participation, environmental observation, and independent judgment.

By scheduling resource demands for terminal devices and consumable resources on the cloud and MEC, an agent carries out the formulation and implementation of the computing task-offloading plan, ensuring the user's highest level of service. The rate of cloud and MEC resource utilization is increased, and energy consumption is decreased [9–11].

Partitioning is a good way to transfer tasks to a cloud or cloudlet; however, there is a huge challenge with how to partition effectively and efficiently. There have been several solutions to this problem offered in the literature. Various schemes have varying benefits and drawbacks depending on how long they take to execute and how much energy they use. This study suggests two new algorithms as an effective solution to this issue. We also have a way to address the offloading issue. The main contributions to the work are as follows.

1. In the first one, a novel multistage graph partitioning idea is implemented in this paper to differentiate the different tasks for processing at each level of the graph.
2. The second algorithm selects locations for offloading tasks between cloudlet and cloud accordingly. The proposed approach uses a three-tier architecture that includes mobile, cloudlet, and cloud, and an agent is active at each level. In this scheme, agents active on three mobile, edge, and cloud servers interact with each other and contain the updated status of all active entities. Agent creates real-time computing offloading strategies for MEC and MCC based on the resource conditions of mobile, edge, and cloud servers.
3. Based on variables including energy consumption and execution time on different bandwidths, we evaluate the system's performance. The paper is divided into five categories. Sect. "Related Works" discusses the literature review. Sect. "Mobile Agent Architecture" describes the mobile agent architecture recommended strategy in detail. Sect. "System Model" discusses the system model of the proposed technique. Sect. "Proposed Agent-Based Multistage Application Partitioning Scheme" provide details about the proposed agent-based partitioning framework. In Sect. "Experiment and Result Analysis", we discuss the experimental results, and Sect. "Conclusion" summarizes the proposed approach in conclusion.

Related Works

To solve the difficulties of compute offloading, offloading choice, and application-partitioning frameworks, numerous offloading frameworks and application-partitioning techniques have been suggested in the literature. In addition, due to the limited capacity of the disc, they need computation support from local or cloud resources. Therefore, mobile cloud computing is used to provide computation processing for demanding applications and cover the limited capacity of smartphones [1, 2].

Framework for Computational Offloading

An adaptive code offloading framework based on machine learning algorithms is proposed [3] for optimizing resources on mobile devices. Mobile application methods that require high computation power mark offloading components. The time and energy required for execution for each method are calculated using a machine learning algorithm. The lowest-cost place is chosen for the execution of methods among mobile, local PC, and the cloud. There is a significant reduction in the execution time and energy consumption of applications. In [5, 6], authors proposed a multi-agent system for offloading based on machine learning. The location for execution is determined by the cost of the tasks; agents apply an online machine learning algorithm. They find supervised learning better than reinforcement learning when contexts are defined by more attributes. To [12] save energy and meet deadlines, a simple min-cut algorithm method is described to decide which component should be offloaded to the cloud. In [13], authors describe how to handle subtask scheduling in processing as a directed acyclic graph (DAG) to improve the multi-objective optimization problem (CMOP). By combining local and edge processing in parallel, the suggested method significantly lowers latency and energy consumption. The disadvantage is that the scheme divides tasks into local and offloading tasks before execution, which acts as static partitioning.

In [14], authors provide a method that uses two servers in turn for first upload round (FUR) and second upload round (SUR) offloading, both of which are backed by a decision engine system. In this scheme, they assign work to another server in case the primary server is occupied. An efficient task-offloading technique for fog environments is presented in [15]. The model allocates tasks to the nodes in the fog in an effective manner and uses an interface to get data from all of the fog nodes. The work's goal is to reduce the system's overall cost in terms of its makespan and energy usage. In [16] mRARSA, a multi-criteria decision algorithm is implemented to evaluate the performance of architecture. The run-time offloading decision of code depends upon both application task size and device context (network constraints). Two types of algorithms are used in this paper. The first fuzzy-MCDM technique used in the offloading decision algorithm to choose alternative resources. The second is a resource switching algorithm that assigns the resource request according to criteria of preferred choice. The [10] writers used an offloading approach based on particle swarm optimization to lower the energy usage of a sequence of tasks that was network- and deadline restricted. In [17], game strategies for partial compute offloading for multi-user, multi-wireless channels are given. Customers who are worried about battery consumption and execution time, as well as those who are just concerned with execution time due to a sufficient battery, are taken into consideration. Authors in [18] explore energy-efficient resource allocation on a multi-user mobile edge computing platform with limited processing time and bandwidth.

They determine the optimum offloading strategy using dynamic programming.

The authors in [19] proposed architecture of agent-based mobile cloud computing where an agent is employed to carry out decision-making for offloading and terminal devices may quickly get offloading results. To find the best offloading technique, a dynamic programming algorithm was used, which enabled the agent to answer more quickly.

The objective of [20]'s study of the mutual gain interaction of work offloading between two sets of agents was to decrease overall energy consumption. An agent-to-agent task-scheduling process was then used, and the effectiveness of the suggested mechanism was then confirmed by a simulation experiment. Using mobile agent-based application program partitioning, a dynamic performance optimization framework was proposed by [21], which is used for mobile cloud computing. Application programs are dynamically divided between the mobile network and the platform, depending on the availability of resources. To increase the durability of vehicle offload transmission, the authors suggested an efficient redundant offloading technique and utilized the deep Q-learning approach to develop the best task- offloading strategy [22].

Application Partitioning Algorithms

In the literature, researchers have broadly addressed the problem of application partitioning in mobile cloud computing and designed different algorithms to resolve this issue. An [4] elastic MCOP algorithm is proposed based on partitioning cost models to make a suitable application partitioning. It starts with profiling and then builds the weighted consumption graph (WCG). The application graph is re-partitioned if the execution bandwidth changes. In [7], a novel hybrid model is proposed that combines the optimization partitioning algorithm with a machine learning algorithm for effective offloading. An object dependency graph of an application by collecting information from a mobile device is made. The graph was partitioned using a novel partitioning algorithm that divides code into offloadable and unoffloadable parts. An online machine learning algorithm is used to send offloadable parts to the cloud. In [8, 23], authors proposed models for partitioning graphs. If there are any changes in network status, it will repartition the graph, but these models do not consider server status or other changes in context to do repartitioning. Another model that considers network bandwidth changes to update the partitioning is proposed in [24] and gives a new set of partitioning algorithms that work on stateful data streams. To meet the user-defined goals while minimizing the energy consumption of mobile devices, a novel ready-time partitioning method is proposed. This technique works in two phases: in the first phase, the user assigns a deadline to each task before starting the application; in the second phase, the algorithm partitions each task when it is ready to offload. This technique considers server and network bandwidth changes for the partitioning of tasks [9]. Due to the increased complexity of workflow applications, just a portion of DAG has been the subject of various researches [11, 25]. The issue of partitioning programs has been researched using numerous techniques, both from a single-user and a multi-user standpoint. Consideration of the multi-user context is more pertinent than consideration of a single user in a real-world MCC scenario. Partitioning for multi-user applications has been discussed in [17, 18, 26]. A multi-user setting may allow numerous mobile devices to participate in cloud computing resources. The author proposed an offloading strategy for multiple-consumer multi-cloudlet setups to reduce energy consumption by offloading users' tasks to the optimal cloudlet [26]. To effectively partition and move mobile computation into MEC, such as self-driving vehicles, the suggested method proposes a reducing cost offloading policy that uses convex optimization and a Lagrangian approach. It also reveals the effect of task dependency on time to complete.

A novel graph-based approach is suggested in [27] for MEC workflow applications. This particular approach is capable of handling intricate workflow applications that possess nonlinear structures, such as those that are parallel, selective, and iterative. In the meantime, the graph-based partition approach can be used to find the offloading selection option with the lowest energy usage of the end device within deadline constraints. In [28], an algorithm framework called dynamic application-partitioning workload task-scheduling secure (DAPWTS) was developed that incorporates many schemes, including searching nodes, min-cut algorithms, energy-enabled scheduling, failure scheduling, and secure schemes.

Different from earlier research, we present a dynamic partitioning algorithm in this paper that makes use of the agent-based multistage graph partitioning technique, which partitions the task at every stage of the graph and decides where to execute it. We are capable of taking the environment into account while making decisions in this way. To the best of our knowledge, to save energy and execution time, our suggested approach is the first multistage graph partitioning algorithm that tackles the issue of workflow computation partitioning at every stage of the graph.

Mobile Agent Architecture

A software program that can be launched from a computer into a network and travel among the network's computer nodes is called a mobile agent [29]. It can be run on those systems to do its job in the absence of its owner. The mobile agent will move running processes from mobile to cloud servers in a compute offloading system, allowing the resource-constrained device to handle more data. Offloading computation increases client device computation performance while reducing battery consumption on the end-user device [9].

The mobile agent contains 4 modules that perform different types of activities.

- *Data Collection Module*: This module collects and stores information about mobile (U) contexts like memory, computation, bandwidth, etc. This module also gets different parameters like computation power waiting time of cloudlet and cloud to take decisions about task offloading.
- *Graph Module*: This module involves making a multistage graph of application nodes.
- *Partitioning Module*: After authentication completion, the partitioning module takes the parts of the application and decides about the execution of the parts on the basis of their type and context information about mobile and edge.

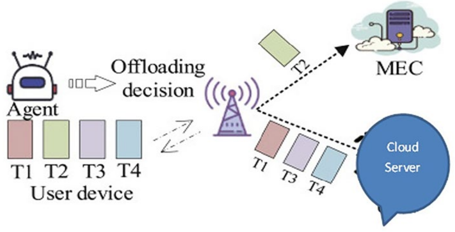


Fig. 1 Agent-based offloading

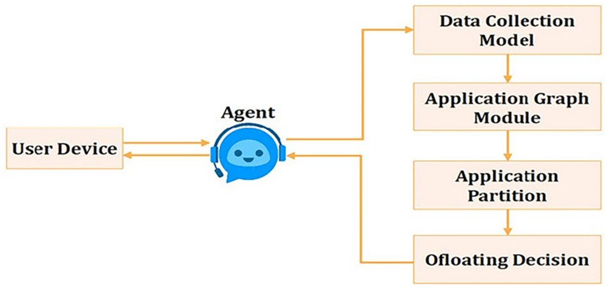


Fig. 2 Agent structure

- *Offloading Module*: This module used to send the off loadable parts to the edge or cloud according to the partitioning decision. This module first checks the cloudlet waiting time and then makes a decision about task offloading.

System Model

The task-offloading architecture of the agent-based MECC is put third. The computing work-offloading issue in MEC and MCC is studied in this section. As seen in Fig. 1, the agent on the mobile device collaborates with resources on the MEC and MCC to provide offloading techniques for the user devices. Offloading the task is considered to be a three-tier task-offloading architecture made up of a cloud server, an edge cloud server, and a user device. The agent creates offloading plans for the user's devices by working with resources on both the MEC and the MCC. The T2 is offloaded to the edge cloud server through the agent's offloading decision when the user requests the offloading of four tasks, and the other three are offloaded to the cloud. A task- offloading architecture is made up of one user device, one edge, and one cloud server (Fig. 2). The agent, using the Modify Ready Time Partitioning (MRTP) algorithm, chooses the location of task offloading after receiving a request for it from the user. We primarily introduce the framework model formulation from two perspectives in this section: problem formulation, time delay, and energy consumption model.

Application Workflow Model

Each individual mobile device user k is denoted by the letter U^K, K are the total number of mobile users, i.e., $1 < k < K$. Here $W^k = (T^k, R^k)$ is a directed acyclic graph with $\left(T^k\right)\left[t^k \dots t^{k_n}\right]$, and t_j^k are sets vertices and edges, used to define an n-task workflow and dependencies on one another for each U^K. In this workflow, tasks are represented by set of vertices $\left(T^k\right)\left[t^{k1} \dots t^{k_n}\right]$, and dependencies between these tasks are represented by R^k, i.e., $\{r_{ij}^k | t_i^k, t_j^k \in T^k\}$. the completion of task s_j^k, and the transfer of the output data to task t_j^k are prerequisites for job t_j^k start, as represented by each dependence r_{ij}^k. Each work $t_i^k T^k$ is characterized as a 2-tuple $t_i^k = (wl^{ki} typek_i)$, where wl^{ki} denotes the size of workload (in million cycles) and $typek_i$ denotes if the task has any restriction about execution place. For each $taskt_i^k$, parameter $type_{ki}$ consider three values in this paper.

- *Unoffloadable*: In applications, some tasks, e.g., cameras, GPS, and user interfaces, need internal component access, or due to some security reasons, they're restricted to execution on mobile.
- *Offloadable*: Maximum numbers of tasks are categorized in this group. These types of tasks have no restrictions about execution place and can start on mobile as well as sent to the cloud according to environmental conditions.
- *Must-offload*: There are some tasks that need some data from cloudlet or cloud side so these types of tasks have restrictions on offloading and cannot start on mobile.

Each task T_j^k needs some input data from T_j^k in kilobytes represented as r_{ij}^k, d_{ij}^k. It contains two fictitious nodes, t_0^k and t_{n+1}^k whose workloads are zero and their types are unoffloadable, to represent the start and completion points of application execution on the mobile device [9].

Cost Models

The cost of computation and communication related to workflow execution is examined in this section. The offloading strategy of tasks in the U^k. application process is stored in the array $S^k = \{s^{k1}, s^{k2}, \dots, s^{kn}\}.s^{ki} = 0$ indicates local execution, whereas $s^k i = 1$ indicates cloudlet execution for the ith task of U^k. Different formulae are used to compute the mentioned charges depending on the task's execution location.

Time Consumption

Computation time of task T_i^k is denoted by $ET(t_i^k)$ with offloading strategy s_i^k calculated by [9].

$$ET\left(t_i^k\right) = \begin{cases} \frac{wl_i^k}{f_m^k} & s_i^k = 0 \\ \frac{wl_i^k}{f_c} + T_{wait} & s_i^k = 1. \end{cases} \tag{1}$$

The clock frequencies of the mobile device and cloudlet in U^k are f_m^k and f_c, respectively. T_{wait} shows how long the task has been in the cloudlet's queue, which is acquired from the cloudlet, and is fully dependent a timely task t_i^k will be run [9].

$$TT(r_{ij}^k) = \begin{cases} 0 & (S_i, S_j) = (0,0), (1,1) \\ L^k + \frac{d_{ij}^k}{dr_{up}^k} & (S_i^k, S_i^k) = (0,1) \\ L^k + \frac{d_{ij}^k}{dr_{up}^k} & (S_i^k, S_i^k) = (1,0), \end{cases} \tag{2}$$

where d_{ij}^k, dr_{up}^k, and L^k depending on the user's existing position in the U^k and its best accessible network, computed dynamically at run-time.

Energy Consumption

The energy spent by task t_i^k execute on place S_i^k denoted by $EE(t_i^k)$, calculated by [9]

$$EE\left(t_i^k\right) = \begin{cases} \frac{wl_i^k}{f_m^k} \times P_{exe}^k & S_i^k = 0 \\ (\frac{wl_i^k}{f_c} + T_{wait}) \times P_{idle}^k & S_i^k = 1. \end{cases} \tag{3}$$

P_{exe}^k and P_{idle}^k represent the each U^k mobile device's operation and standby power, accordingly. $TE(r_{ij}^k)$ is the amount of communication energy consumed by a mobile device belonging to the U^k. It represents the amount of energy used to transmit data between tasks t_i^k and t_j^k using offloading strategies s_i^k and s_j^k, as computed by [9].

$$TE\left(r_{ij}^k\right) = \begin{cases} 0 & (S_i, S_j) = (0,0), (1,1) \\ L^k \, x \, P_{idle}^k + \left(\frac{d_{ij}^k}{dr_{up}^k} \, x \, P_{up}^k\right) & (S_i^k, S_i^k) = (0,1) \\ L^k \, x \, P_{idle}^k + \left(\frac{d_{ij}^k}{dr_{up}^k} \, x \, P_{dl}^k\right) & (S_i^k, S_i^k) = (1,0), \end{cases} \tag{4}$$

$$E^k \sum_{t_i^k} \in T^k EE\left(t_i^k\right) \sum_{t_i^k,} TE\left(r_{ij}^k\right), \tag{5}$$

where P_{up}^k and P_{dl}^k are the download and upload capacity of every U^k mobile phone.$E^k(S^k)$ shows the total energy consumption U^k of mobile device as a result of performing his/her workflow using offloading technique S^k. Clearly, $E^k(S^k)$

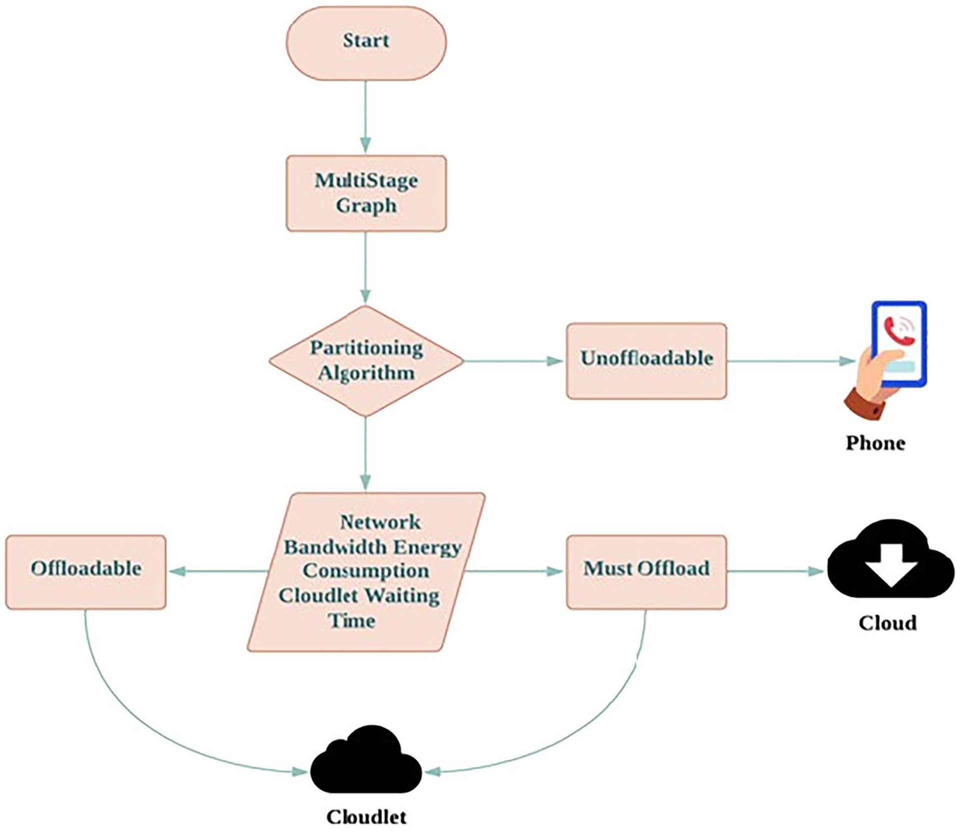


Fig. 3 Flow chart of the proposed multistage partitioning algorithm scheme

is the total amount of energy consumed by all of the jobs in his/her workflow, as well as all of the interactions between tasks.

Formulation Problem

The objective of this article is to properly choose between mobile, cloud, and cloudlet computing for each user's flow processes to lower smartphone energy consumption while still meeting deadline constraints. Therefore, the problem of efficiency in terms of energy restriction in time partitioning may be described as a constrained challenge with workflow Eqs. (6), (7) scheduling [9]. Each U^K makes effort to resolve the issue below.

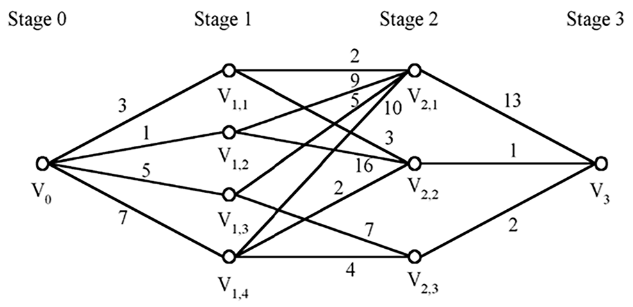


Fig. 4 Multistage graph

$$\min E^k\left(S^k\right). \tag{6}$$

Line with

$$T^k\left(S^k\right) \leq D^k. \tag{7}$$

Proposed Agent-Based Multistage Application Partitioning Scheme

The suggested technique improves efficiency in terms of energy consumption and execution speed by combining partitioning and offloading techniques. To make partitioning effective, the proposed scheme uses a multistage graph idea in which tasks are divided into offloadable, unoffloadable, and must-offload types of arrays at every stage of the graph. Execution decisions about each array of tasks are then subsequently made. In the context of offloading, the proposed method will deploy an active agent in three distinct places: the cloud, the cloudlet, and the mobile device. These agents share information about their environment, gather data about resources that are available, and, on the basis of this information, take decisions regarding offloading tasks on the mobile side. Once location is determined through a partitioning algorithm for each task, the tasks are sent to cloudlet, cloud, or inside mobile for execution. This involves the selection of a suitable location for processing. Flow chart in Fig. 5 presents the overview of our proposed approach (Fig. 3).

Partitioning Using Multistage Graph

The main component of our suggested partitioning solution is the conversion of the application program into a graph module. The multistage graph of the application is created in this step by the agent's graph module and is used as input for the partitioning stage. The goal of using a multistage graph is to make decision-making easier at every level of the graph. The approach introduced in [4] involves utilizing the (WGC) for an application, which serves as input for a partitioning algorithm. This algorithm then determines the allocation of all nodes within the graph based on contextual parameters. But it has the drawback that it decides about all nodes in the application graph collectively and then merges offloadable nodes into a single node, which is then transmitted to the cloud. This is ineffective, especially when bandwidth is restricted. In the same way, the method described in [9] used a directed acyclic graph as the partitioning algorithm's input. In this approach, each task of the graph is individually processed in real time as it becomes ready for execution. However, it takes a lot of time to handle each task separately within the graph. To address this challenge, we propose an innovative multistage graph partitioning technique. At each stage of the graph, tasks are categorized into three arrays: unoffloadable, offloadable, and must-offload. Subsequently, the offloading algorithm makes decisions at each stage based on the task arrays and contextual parameters. Figure 4 illustrates a multistage graph with numerous nodes or tasks at each stage of the graph.

Algorithm 1: Algorithm 1

```
Step 1: Create three arrays a, b, and c
$Array[a]$
$Array[b]$
$Array[c]$
Step 2: Agent makes the weighted consumption multistage graph of application G with N nodes.
$Input(A) \leftarrow G$
Step 3: Multistage graph contain different stage with different number of nodes.
$Stages(s) = \{s1, s2 \dots sn\}$
Step 4: Each stage contain different number of nodes
$S1 = \{n1, n2 \dots nn\},\ S2 = \{n1, n2 \dots nn\},\ sn = \{n1, n2 \dots nn\}$
Step 5: Input: $S\{s1, s2 \dots sn\}$
Output: $Sk\{s1[a, b, c], s2[a, b, c] \dots sn[a, b, c]\}$
Step 6: While there exist unscheduled stage S
While there exist unscheduled stage S: Step
7: Take each stage one by one $S[] == s$
Step 8: Call Multistage Algorithm2 on stage S
        $Sk$ call Multistage Algorithm2(S)
```

Algorithm 2: Multistage Algorithm2(S)

```
1: If n is unoffloadable, store it to array $a$
2: If $(n)$ == $Unoffloadable$:
3: $[a]$ == $n$
4: If $n$ is mustoffload, store it to array $b$
5: If $(n)$ == $Mustoffload$:
6: $[b]$ == $n$
7: If $n$ is offloadable, store it to array $c$
8: If $(n)$ == $Offloadable$:
9: $[c]$ == $n$
```

This is a brief summary of algorithms 1 and 2: Make a triple array of a, b, and c. For an application with N nodes, the agent creates a weighted consumption multistage graph G. With a different number of nodes in each level, specify the stages of the multistage graph by entering graph G as an input. Describe the phases of the multistage graph. Every level has a different number of nodes. Repeat these steps to cycle through each stage: a node that is not able to be offloaded is stored in array b. It will add to array b if it has to be must-offload. Place a node in array c if it is designated as offloadable. The method proceeds with the process for each unscheduled phase in the multistage graph. When the algorithm has finished iterating over each stage of the multistage graph, all of the nodes in the graph will be stored in the arrays a, b, and c, directed according to their respective natures.

Offloading Decision Phase

This phase largely focuses on determining the execution location of each task array at every stage of the multistage graph while taking into account the immediate environment, such as cloudlet server resources and network state. It seeks to reduce the mobile device's energy consumption. The partitioning module of the agent is active and executes the offloading algorithm in this phase to choose the best location for each array of tasks.

Algorithm 3: Algorithm 3

```
1: Input: $sk\{s1[a,b,c], s2[a,b,c] \ldots sn[a,b,c]\}$
2: Output: offloading strategy (OS) for Unoffloadable, Offloadable and Mustoffload
   arrays at each stage
3: While
4: There exists un-partitioning stage $sk$
5: Do
6: Take $sk$ from $s1$ to $sn$
7: $OS$ call Algorithm 4
8: End
```

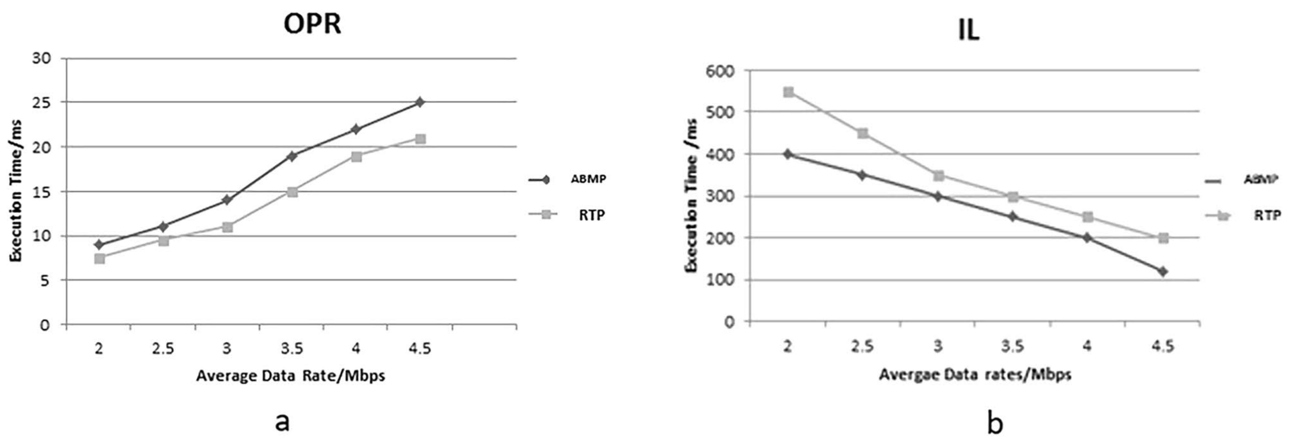


Fig. 5 Execution time comparison for OPR and IL

Algorithm 4: Decision about different arrays

```
Input: Array[a], Array[c], Array[b]
Output: Array [computed]

Step 1: Compute Array[a] on the mobile device.
Step 2: Identify the offloadable nodes in Array[c].
Step 3: Get the current cloudlet capacity and bandwidth from the profiler module.
Step 4: Calculate the total time required to compute Array[c] on the cloudlet, including the network bandwidth cost.
Step 5: If the total time is less than or equal to the time required to compute Array[c] on the mobile device:
Step 6: Send Array[c] to the cloudlet.

Step 7: Else if the total time is greater than the time required to compute Array[c] on the mobile device, but the bandwidth is sufficient to send Array[c] to the cloudlet:
Step 8: Compress Array[c] using Huffman coding.
Step 9: Send the compressed Array[c] to the cloudlet.
Step 10: Else:

Step 11: Divide Array[c] into two parts.

Step 12: Send each part of Array[c] to the cloudlet one by one.
Step 13: Compute Array[b] on the cloudlet.

Step 14: Merge the computed results of Array[a], Array[c], and Array[b] into Array [computed].
Step 15: Return Array [computed].
```

The final outcome of the method is the offloading strategy (OS) for the input set's unoffloadable, offloadable, and must-offload lists at each level. The algorithm proceeds as follows, while the unpartitioning stage is present sk. A decision-making procedure for transferring compute burdens from the mobile to a cloudlet or server in the cloud is described in the code snippet. It entails locating the data array's offloadable nodes, figuring out how much time and network bandwidth are needed, and deciding whether or not to transmit the nodes to the cloudlet. The nodes are transmitted to the cloudlet if the conditions are satisfied. They are compressed and reconsidered, if not. The nodes are separated and transmitted separately if the conditions are still not satisfied. A similar procedure is carried out for a different set of nodes. The nodes are transferred to the cloud if there is enough bandwidth; otherwise, they are compressed and delivered. This approach was created to reduce overall calculation time while accounting for the bandwidth and capacity of the cloudlet and cloud.

Experiment and Result Analysis

This section will analyze the effectiveness of the suggested offloading method in terms of throughput and execution time.

Experiment Setting

To demonstrate the feasibility and efficiency of the offloading strategy, we conducted multiple sets of comparative tests. In the experiment, the amount of energy used and the delay in task execution caused by the user's task request were evaluated. A multistage graph of application was made by SOOT [30]. Experiments were conducted on three applications, object and pose recognition (OPR), indoor localization (IL), and object character recognition (OCR), where one cloud and one edge cloud offered services to a user at some point. The Cloudsim toolkit is used to run the algorithms on a system with an Intel Core i5 processor running at 1.7 GHz and 4 GB of RAM, as well as an Intel Core i9 processor running at 2.6 GHz and 16 GB of RAM, to examine the difference between machine based. For experiments, the same parameter values and application description are used as in Table 1 and Table 2 of [9].

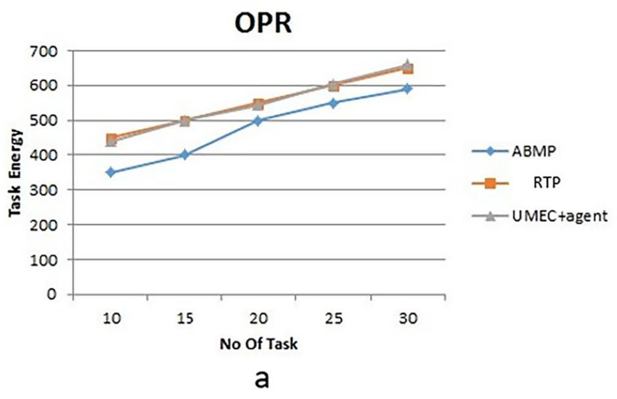
Results and Discussion

In this part, the effectiveness of the D2-RTP [9] and ABMP algorithms is reviewed. The agent takes part in both the planning and implementation of work-offloading schemes. The outcome demonstrates that including an agent in the scheme reduces communication time and improves task throughput. We divide our experiment into two sections.

Impact of Bandwidth Size on Performance

In this step, we conduct an experimental study to determine how bandwidth affects performance. The outcomes of the experiment are displayed in Fig. 5a, b. All other factors in the experiment are constant, whereas variation in the bandwidth depicts how bandwidth affects execution time.

Figure 5a, b makes it clear that, regardless of the varying bandwidth conditions, the ABMP method consistently has a shorter execution time than the D2-RTP technique. Both partitioning techniques operate with noticeably shorter execution times as the bandwidth rises. This finding suggests that greater bandwidth enables these algorithms to run more effectively, resulting in shorter processing times. Figure 5a, b highlights the importance of bandwidth in enhancing the



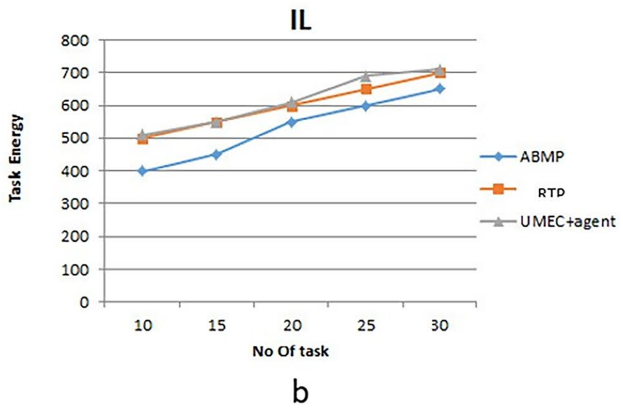


Fig. 6 Energy comparison for different tasks

efficiency of ABMP and D2-RTP since their execution times get shorter as larger data transfer rates become available.

Impact of Number of Tasks

In this section, experiments conduct to compare the proposed ABMP algorithm's performance to two existing algorithms UMEC + agent [19] and D2-RTP used in [9].

Three different algorithms D2-RTP, UMEC+ agent as reported in reference [19], and the recently suggested method ABMP is displayed in Fig. 6a and b used for two different applications, IL and OPR. These graphs show how task energy is produced when computing offloading tasks are being carried out. The difficulty is in accommodating user demands within the allotted task deadlines, primarily through offloading work to edge computing. Notably, when compared to both the D2-RTP methodology and the agent-based UMEC method described in reference [19], the multi-stage partitioning strategy with agent participation, demonstrated in the ABMP algorithm, stands out for its excellent energy efficiency during task execution.

Conclusion

An innovative idea for computing task partitioning and offloading that aims to achieve optimal partitioning and effective offloading is proposed in this research. The article gives an algorithm for task partitioning by using a novel multistage graph idea and offloading by introducing an agent. The core concept of a multistage graph involves making decisions at various graph stages, addressing the constraints of making a single decision for the entire graph task, while also optimizing time compared to individual decision-making for each task. Three-tier architecture with agent support is given for task offloading, in which the agent integrates resources from user devices, the cloud, and the edge to aid in task execution. The results of the experiment showed that ABMP optimally reduces task partitioning, offloading execution times, and energy consumption. In the future, we will suggest a more extensive network model that may establish connections with ad hoc mobile devices, also known as the virtual cloud. Additionally, we will test our solution in a multi-user environment.

Data Availability Data sharing is not applicable to this article as no new data were created or analyzed in this study. However, the data that support the findings of this study are available from the corresponding author, upon reasonable request.

Declarations

Conflict of Interest The authors declare they have no conflict of interest.

References

1. Hassan M, Al-Awady AA, Ali A, Iqbal MM, Akram M, Khan J, AbuOdeh AA. An efficient dynamic decision-based task optimization and scheduling approach for microservice-based cost management in mobile cloud computing applications. Pervasive Mob Comput. 2023;92: 101785.
2. Kavya G, Sureka V, Sudha L, Aruna K. Mobile cloud computing for computation offloading using application partition in algorithms: taxonomy, review techniques. Math Stat Eng Appl. 2022;71(32):535–49.
3. Nawrocki P, Sniezynski B, Slojewski H. Adaptable mobile cloud computing environment with code transfer based on machine learning. Pervasive Mob Comput. 2019;57:49–63.
4. Wu H, Knottenbelt WJ, Wolter K. An efficient application partitioning algorithm in mobile environments. IEEE Trans Parallel Distrib Syst. 2019;30(7):1464–80.
5. Nawrocki P, Sniezynski B. Adaptive service management in mobile cloud computing by means of supervised and reinforcement learning. J Netw Syst Manag. 2018;26(1):1–22.
6. Nawrocki P, Śnieżyński B, Kołodziej J. Agent-based system for mobile service adaptation using online machine learning and mobile cloud computing paradigm. Comput Inform. 2019;38(4):790–816.
7. Kaya M, Çetin-Kaya Y. Seamless computation offloading for mobile applications using an online learning algorithm. Computing. 2021;103(5):771–99.
8. Haghighi V, Moayedian NS. An offloading strategy in mobile cloud computing considering energy and delay constraints. IEEE Access. 2018;6:11849–61.
9. Shadi M, Abrishami S, Mohajerzadeh AH, Zolfaghari B. Ready-time partitioning algorithm for computation offloading of workflow applications in mobile cloud computing. J Supercomput. 2021;77:6408–34.
10. Wang Y, Wu L, Yuan X, Liu X, Li X. An energy-efficient and deadline- aware task offloading strategy based on channel constraint for mobile cloud workflows. IEEE Access. 2019;7:69858–72.
11. Kao Y-H, Krishnamachari B, Ra M-R, Bai F. Hermes: latency optimal task assignment for resource-constrained mobile computing. IEEE Trans Mob Comput. 2017;16(11):3056–69.
12. Stoer M, Wagner F. A simple min-cut algorithm. J ACM (JACM). 1997;44(4):585–91.
13. Cui Y-Y, Zhang D-G, Zhang T, Zhang J, Piao M. A novel offloading scheduling method for mobile application in mobile edge computing. Wirel Netw. 2022;28(6):2345–63.
14. Aldmour R, Yousef S, Baker T, Benkhelifa E. An approach for offloading in mobile cloud computing to optimize power consumption and processing time. Sustain Comput: Inform Syst. 2021;31: 100562.
15. Lone K, Sofi SA. Cost efficient task offloading for delay sensitive applications in fog computing system. SN Comput Sci. 2023;4(6):817.
16. Islam A, Kumar A, Mohiuddin K, Yasmin S, Khaleel MA, Hussain MR. Efficient resourceful mobile cloud architecture (mrarsa) for resource demanding applications. J Cloud Comput. 2020;9:1–21.

17. Zhou S, Jadoon W. The partial computation offloading strategy based on game theory for multi-user in mobile edge computing environment. Comput Netw. 2020;178: 107334.
18. Kuang Z, Guo S, Liu J, Yang Y. A quick-response framework for multi-user computation offloading in mobile cloud computing. Futur Gener Comput Syst. 2018;81:166–76.
19. Wang R, Cao Y, Noor A, Alamoudi TA, Nour R. Agent-enabled task offloading in uavaided mobile edge computing. Comput Commun. 2020;149:324–31.
20. Gu B, Chen Y, Liao H, Zhou Z, Zhang D. A distributed and context-aware task assignment mechanism for collaborative mobile edge computing. Sensors. 2018;18(8):2423.
21. Angin P, Bhargava BK. An agent-based optimization framework for mobile- cloud computing. J Wirel Mob Netw Ubiquit Comput Depend Appl. 2013;4(2):1–17.
22. Zhang K, Zhu Y, Leng S, He Y, Maharjan S, Zhang Y. Deep learning empowered task offloading for mobile edge computing in urban informatics. IEEE Internet Things J. 2019;6(5):7635–47.
23. Kaya M, Koçyiğit A, Eren PE. An adaptive mobile cloud computing frame- work using a call graph based model. J Netw Comput Appl. 2016;65:12–35.
24. Ding S, Yang L, Cao J, Cai W, Tan M, Wang Z. Partitioning stateful data stream applications in dynamic edge cloud environments. IEEE Trans Serv Comput. 2021;15(4):2368–81.
25. Liu T, Chen F, Ma Y, Xie Y. An energy-efficient task scheduling for mobile devices based on cloud assistant. Futur Gener Comput Syst. 2016;61:1–12.
26. Mazouzi H, Achir N, Boussetta K. Dm2-ecop: an efficient computation offloading policy for multi-user multi-cloudlet mobile edge computing environment. ACM Trans Internet Technol (TOIT). 2019;19(2):1–24.
27. Li X, Chen T, Yuan D, Xu J, Liu X. A novel graph-based computation offloading strategy for workflow applications in mobile edge computing. IEEE Trans Serv Comput. 2022;16(2):845–57.
28. Lakhan A, Li J, Groenli TM, Sodhro AH, Zardari NA, Imran AS, Thinnukool O, Khuwuthyakorn P. Dynamic application partitioning and task- scheduling secure schemes for biosensor healthcare workload in mobile edge cloud. Electronics. 2021;10(22):2797.
29. Chess D, Harrison C, Kershenbaum A. Mobile agents: are they a good idea?—update. In: Vitek J, Tschudin C (eds) Mobile object systems towards the programmable internet: second international workshop, MOS'96 Linz, Austria, July 8–9, 1996 Selected Presentations and Invited Papers 2. Berlin, Heidelberg: Springer; 1997. pp. 46–47.
30. Einarsson A, Nielsen JD. A survivor's guide to java program analysis with soot. BRICS, Department of Computer Science, University of Aarhus, Denmark 2008; 17