GUB Journal of Science and Engineering (GUBJSE)
p-ISSN: 2409-0476
vol. 09, no. 01, 2022

Task Offloading and Execution in Edge-Cloud Collaborative Network Using Genetic Algorithm

Jargis Ahmed[1]

1 Department of CSE, Green University of Bangladesh (GUB), Dhaka, Bangladesh

*Corresponding author's email: jargis@cse.green.edu.bd

Received: 22/05/2022
Accepted: 27/09/2022
Published: 13/07/2024

Data Availability: The data are available on request from the corresponding author.

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

DOI: 10.3329/gubjse.v9i1.74883

Abstract

Mobile Edge Computing (MEC) system is now one of the most emerging sectors in wireless communication. It provides computation capability near the edge users and reduces the dependency on the Internet Cloud (IC). The edge users offload the task to the MEC server due to the lack of computing resources for executing resource-hungry applications. MEC servers can serve faster as they are quite close to the edge users but have limited computation resources compared to IC. On the other hand, IC has abundant computation resources but offloading to IC will add extra latency to complete tasks execution. In this scenario, an MEC has to decide intelligently which tasks should be executed by itself and which should be sent to IC. In this paper, we addressed this issue of task assignment and execution either on MEC or on IC and formulated an optimization problem to reduce the task processing latency. The problem is Integer Linear and NP-Complete, thus having higher time complexity. In this regard, we proposed a Genetic algorithm-based Meta-Heuristic method to solve the task completion problem considering the delay constraint of the user. Simulation results of the proposed method indicated improvement in terms of successful task execution, task turnaround time, and better Quality of experience (QoE) compared to the state-of-the-art works.

Keywords: Mobile Edge Computing, Task Assignment, Genetic Algorithm, Quality of Experience, Internet Cloud.

Highlights

- Efficient task assignment in Edge-Cloud Networks using Genetic Algorithm.
- Improved task execution success rate and turnaround time.
- Enhancing Quality of Experience (QoE) for edge users through intelligent task offloading.

Acknowledgements

This work was supported in part by the Center for Research, Innovation, and Transformation (CRIT) of Green University of Bangladesh (GUB).

1 Introduction

The usage of mobile devices is increasing and the demand for computing capability is getting higher at the edge network. Resource-hungry applications such as augmented reality, online gaming, natural language processing, etc. [1]–[3] getting popular. However, the processing capability and the requirements of external computation resource initiates the task offloading from the user device to the cloud server. As the communication latency with the cloud server is higher, the Mobile Edge Computing server (MEC) can play a significant role in decreasing the delay [4], [5]. Also, the MEC servers can share their resources with each other to increase the computation capability at the edge network [6]. The user equipment (UE) has a delay constraint for executing latency-critical tasks. A UE can be any mobile device like a smartphone, laptop, IoT device, etc. When external computation is required for certain applications then the UE offloads data to the MEC server. The MEC server can serve UEs faster than cloud servers as they are deployed at the edge network. Although MEC is faster in terms of serving UE, they are not resource-rich like cloud servers. Hence, they cannot serve all the tasks offloaded by the UEs at a time. Again, if a UE offloads tasks directly to IC, this will add extra latency for communication and data transmission. In that case, if UEs offload tasks to the MEC server, then due to resource shortage tasks will be placed in the queue for execution, and hence queuing delay will be increased. To aid the situation and to meet individual task delay constraints, MEC can collaborate with the cloud server. MEC server can send some tasks to IC to reduce queuing latency. Now the question will arise, which tasks should be executed on MEC, and which tasks should be executed on the IC? The MEC server has to solve this problem intelligently to maintain a better Quality of experience (QoE) for UEs.

In the paper [7] authors have considered an environment where UE offloads tasks to MEC and they formulated this offloading problem using game theory and proved Nash equilibrium among mobile devices while offloading happens. Again, after observing [8], [9] and [10], we saw some lack while providing computation resources for executing tasks offloaded by the UEs. They barely consider the quality of experience (QoE) of the UEs. To achieve QoE, a collaboration between MEC and IC is required. In this process, MEC with fewer resources can offload some of the incoming tasks to the IC which holds enough abundant resources. In [11] authors proposed an IoT-based smart stick that can be connected to an MEC server; thus, we can consider the tasks offloaded by that device will be latency critical. When we are dealing with latency-critical tasks, the challenging problems here are what to offload and what should not, when to offload, and how much should be offloaded to the IC. In [12] authors proposed collaboration of MEC and IC in the telemedicine or smart health care environment. They tried to reduce the offloading cost to the hospital of the UEs. They proposed a genetic algorithm-based solution called CEGA where, MEC directs UEs to offload tasks either to MEC or IC without considering task execution latency. In [13] authors considered MEC and IC and proposed a cooperative scheduling scheme (CSS) which is a binary decision-making process to offload tasks. In their approach when the MEC server is going to face resource scarcity due to the queued tasks, MEC decides to offload newly incoming tasks directly to the IC server without considering the task completion delay deadline. On the other hand [14] proposed an execution delay-aware greedy method called heuristic tasks assignment (HTA) for completing tasks execution either in MEC or IC server. They calculated a ratio of the task completion time requirement of MEC and IC and used this as a heuristic value. Based on the heuristic, tasks are separated for execution on MEC and IC. In this case, the solution may not be reliable as this approach does not consider all possible cases to generate a solution.

In this paper, we have designed tasks offloading and execution strategy collaborating MEC and IC. In this method, the tasks offloaded from UEs first come to the MEC server. The MEC server will be responsible for partitioning tasks into two groups. One group of tasks will be executed in MEC and the other in the IC server. The MEC server must partition the tasks intelligently based on the delay constraints of the tasks and the computation and storage resource availability of the MEC server. Hence an optimization problem needs to be formulated and solved by the MEC server to minimize overall task completion time. After observing the complexity of the proposed optimization problem, we have designed a meta-heuristic solution called Genetic Algorithm-based Task Assignment (GATA). Here we designed each chromosome of the initial population based on the decision factor of the incoming tasks. The decision factor is a binary variable that indicates a task will be placed either in MEC or in IC for execution. The value of this factor will be randomly selected. Now crossover and mutation process will generate new chromosomes and the next generation will be created. After running several iterations, the best chromosome will be selected based on the fitness value. From this chromosome we can easily create two disjoint task sets (based on the value of the decision factor) to execute tasks in MEC and IC. The major contribution of our proposed policy is summarized below:

- We formulated our problem of task assignment on MEC and IC server as a delay optimization problem.

Where tasks offloading delay, execution delay, and queuing delay are considered. The objective is to complete task execution within the task delay deadline, and resource constraints of the MEC server to enhance better QoE of the UEs

- The optimization problem is an Integer Linear and NP-complete problem. Although the optimal solution is highly time-consuming according to [14].
- We proposed a meta-heuristic algorithm GATA to reduce the time complexity of the optimization problem. Here we iterate the algorithm for several hundreds of generations to achieve a near-optimal solution.
- Finally, we compared our work with some related literature work CEGA [12], HTA [14] and CSS [13]. The results represent the performance enhancement of our proposed policy compared to the existing works.

The rest of the paper is organized as follows. Section 2 contains our present literature study. In Section 3, we elaborate on our system model and assumptions. Problem formulation and solution are designed in Section 4. In Section 5, we described and analyzed the simulation results. Finally, we conclude our paper in Section 6.

2 Related works

In recent years, usage of the mobile edge computing server has increased dramatically. As most people are using smart devices or making smart homes using IoT and Internet connectivity, the necessity of external computation is increasing. Hence the devices offload tasks to nearby MEC servers to execute their tasks. In [7], Chen et al. discussed the offloading process where the decision to offload is taken by mobile devices by running a game theoretic algorithm that achieves Nash equilibrium. Even though the algorithm was slightly worse than the centralized method. Their main drawback was the consideration of unlimited resource capacity in the telecom (MEC) cloud. This is not practical as MEC servers are designed as small-scale servers [15]. In [8] Liu et al. encountered a challenging two-timescale stochastic optimization problem of scheduling tasks offloaded by UEs to the MEC server. The key limitation of the proposed method is that the mobile devices require feedback from the MEC server to decide whether to offload or not and eventually it results in decreasing QoE of the user. Load sharing scenario between local cloud and Internet Cloud discussed in [9] where Gelenbe et al. considers average response time and energy consumption of the servers. Although average delay measurement sometimes provides an inappropriate conclusion for burst traffic. In [10] Guo et al. proposed a different design policy that assigns tasks generated at the mobile subscribers with edge clouds according to the index which is calculated by considering edge cloud delay and operational power consumption. But again, consideration of the limited resource constraints at the edge server was not there and energy at the edge server should not be that much of a concern.
In [12], authors designed a task offloading scheme considering the telemedicine environment. Here they designed a smart healthcare system called Cost Effective Genetic Algorithm for Tasks Offloading (CEGA) where UEs are patients, and they can offload the health status to the hospital using MEC or IC. They optimized the cost of offloading the UEs. They only considered the tasks offloading choice of the UEs and focused only on minimizing the cost of UEs for offloading tasks to remote servers combining latency and energy cost. The impact of task execution and the queuing delay of MEC and IC servers is not considered. In [13] Zhao et al. discussed a cooperative scheduling scheme (CSS). They designed a threshold-based scheduling scheme knowing the limited resource availability of the MEC server, thus they collaborated with the Internet cloud's abundant resources to serve incoming tasks. The limitation of their work is they offload newly incoming tasks directly to the IC server if the tasks queue of the MEC is full. The authors in [14] designed an optimal task scheduling policy considering task execution delay. Although the complexity of the problem is high. Hence, they proposed a heuristic task assignment (HTA) algorithm for partitioning tasks for executing in MEC or IC. Again, the heuristic greedy approach could stuck in the local optimum solution and can overlook the best possible solution for task placing either in MEC or IC server and completing the execution considering offloading delay, queuing delay, and execution delay.

3 System Model

Fig. 1 represents the total system where we are considering Mobile device users as UE, MEC server consists of multiple VMs and is connected to the Internet cloud. The UEs are scattered and distributed according to the Poisson Point Process. We assumed that there is an n number of UEs and that all the UEs are connected to the MEC server by the wireless medium.
The task arrival rate at the MEC server is λ and we considered each user will offload one task at a certain reference time. The MEC server has limited computation and storage resources, on the other hand, we considered

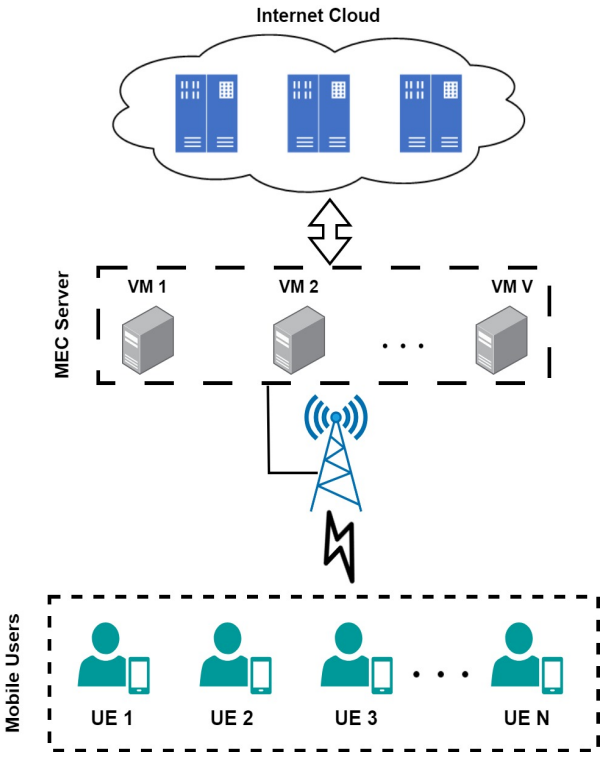


Fig 1. System Architecture

the Internet Cloud (IC) to have abundant computation resources compared to MEC. When UE cannot be able to compute the tasks by itself, then the tasks are offloaded to the MEC server. MEC servers having limited computing resources will determine which tasks can be executed and which tasks should be sent to the IC based on the task delay constraints and resource availability constraints of the MEC server. After the execution of the tasks, the MEC server will provide the results back to the UEs. The list of notations of variables we have assumed and used to express our equations is given in Table 1.

3.1 Network Model of UE and MEC

UEs are executing delay-sensitive applications and if they cannot compute a task and get the desired result by themselves then tasks will be offloaded to the MEC via wireless communication using the base station. Let's assume a UE i generates a task $\mathcal{A}_i = (\mathcal{D}_i, \mathcal{C}_i, \mathcal{T}_i)$ where $\mathcal{D}_i$ is the the task data size in bits, $\mathcal{C}_i$ is the number of CPU cycles required to process per bit of the task, and $\mathcal{T}_i$ is the task delay deadline. Also, suppose the MEC allocates bandwidth for a UE i is the B_i where the total bandwidth is B. As the channels

Table 1. Notations of variables

Description	Notations
Task Delay Deadline	$\mathcal{T}$
Task Data Size	$\mathcal{D}$
MEC Turnaround Time	T_e^i
Internet Cloud Turnaround Time	T_c^i
Task Arrival rate	λ
Number of VM/Servers in MEC	S
Queuing Time in MEC	q
Bandwidth	B
Total tasks set at MEC	$\mathcal{S}_n$
Tasks set for MEC execution	$\mathcal{S}_e$
Tasks set for IC execution	$\mathcal{S}_c$

can be noisy, the effective data rate R_i can be calculated as

$$R_i = B_i + \log_2\left(1 + \frac{p_i h_i}{\rho^2}\right), \tag{1}$$

where the transmission power is p_i, channel gain is h_i of i_{th} user. Also, ρ^2 represents the noise power which is normalized by the transmission power p_i. Therefore, the delay for offloading is

$$T_i^o = \frac{\mathcal{D}_i}{\mathcal{R}_i} \tag{2}$$

In this paper, we ignored the delay in downloading the result after completing task execution, as the downlink data rate is considerably much higher [16] compared to the uplink data rate.

3.2 MEC Computation and Storage Resource Model

MEC server has limited computation resources, where the VM is the smallest unit that can execute tasks offloaded by the UEs. Suppose the MEC has a total $\mathcal{W}$ GHz computing resource, $\mathcal{Q}$ GB of storage resource, and $\mathcal{V}$ number of VMs. Hence the computation ability of VM is $\mathcal{W}/\mathcal{V}$. For a UE i, as CPU cycles required for processing per bit is $\mathcal{C}_i$ then CPU cycle required for executing tasks $\mathcal{A}_i$ is

$$\omega_i = \mathcal{C}_i \times \mathcal{D}_i. \tag{3}$$

If the computation resource of the VM where task $\mathcal{A}_i$ will be executed is f_i^e then the execution time at MEC for the task will be

$$T_i = \frac{\omega_i}{f_i^e} \tag{4}$$

Now the total delay faced by the UE i for executing the task

$$T_e^i = T_i^o + T_i + q_i \tag{5}$$

where q_i is the queuing delay faced by the tasks $\mathcal{A}_i$ at MEC server queue. We considered the MEC server has a task queue that follows the M/M/C model according to the paper [14], hence for the case of simplicity we considered the equations for calculating queuing delay from [14].

3.3 Internet Cloud Model

The MEC server is connected to the Internet cloud via the wired backhaul network. After offloading a task to the MEC server by a UE i, the MEC will determine if the task can be executed in that server, or if it will require sent to the IC. If there is a resource shortage and the task delay constraints permit, then the tasks should be executed in the IC. Hence the UE must face execution, transmission, and reception delays. We represent this IC delay using T_c^i following the proposed model of [13], [14].

4 Problem Formulation and Meta Heuristic Solution

In this section, we have formulated our optimization problem and based on the nature of the problem's complexity, a meta-heuristic solution approach is.described.

4.1 Optimal Tasks Assignment

As the tasks have delay constraints and should be executed, and the result of execution needs to be feedback within the delay deadline. The MEC server needs to decide which tasks should be executed in it and which tasks should be executed in the IC. As we consider the task execution delay should be optimized to ensure better QoE of the UEs, we need to find out the optimal task assignment sets for MEC $\mathcal{S}_\urcorner$ and for IC $\mathcal{S}_\lrcorner$ from the total tasks set $\mathcal{S}_\setminus$. Hence the problem is a minimization problem where we minimize the total tasks completion delay and generate two disjoint task sets $\mathcal{S}_\urcorner$ and $\mathcal{S}_\lrcorner$. Thus, the problem can be illustrated according to the paper [14] using Eq. 6 to Eq. 12 as follows

$$\underset{\mathcal{S}_e, \mathcal{S}_c \in P\mathcal{S}_n}{\operatorname{argmin}} \quad z(T) = \sum_{i \in \mathcal{S}_e} T_e^i + \sum_{j \in \mathcal{S}_c} T_c^j \tag{6}$$

Subject to :

$$T_e^i \leqslant \mathcal{T}_i, \tag{7}$$

$$T_c^j \leqslant \mathcal{T}_j, \tag{8}$$

$$\sum_{i \in \mathcal{S}_e} \omega_i \leqslant \mathcal{W} \tag{9}$$

$$\sum_{i \in \mathcal{S}_e} \mathcal{D}_e^i \leqslant \mathcal{Q} \tag{10}$$

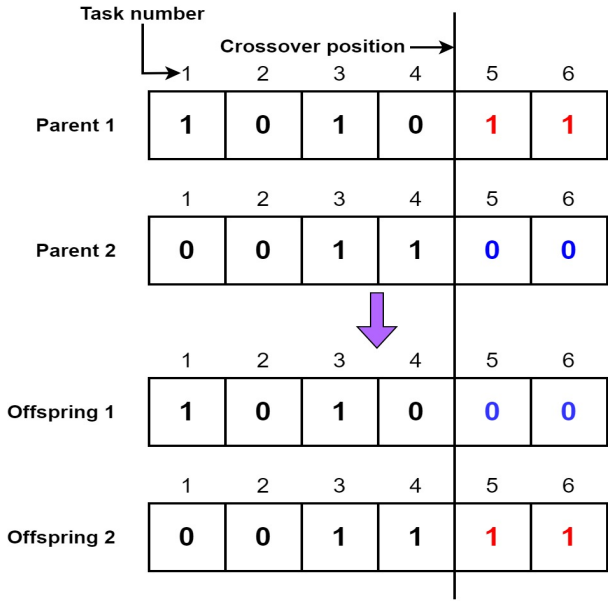
$$\mathcal{S}_e \cup \mathcal{S}_c \subseteq S_n, \tag{11}$$

$$\mathcal{S}_e \cap \mathcal{S}_c = \emptyset. \tag{12}$$

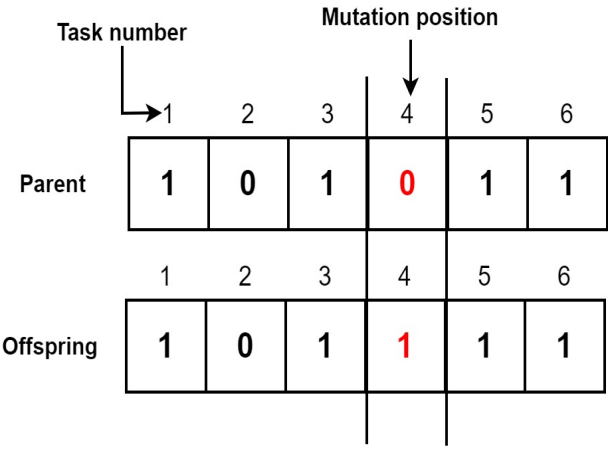
The objective function in Eq. 6 finds a minimum processing delay of all tasks offloaded to the MEC server, where tasks in set $\mathcal{S}_e$ are executed in MEC and tasks in set $\mathcal{S}_c$ are executed in IC.
Eq. 7, 8 refers that the task completion time required by MEC and IC must have to fulfill the delay deadline of that particular task. The total computation and storage resources are fixed for the MEC server, hence constraints (9) and (10) ensure the tasks which will be executed in the MEC server should not exceed the available computation CPU cycle and storage resource respectively. Constraint in Eq. 11 represents that union of $\mathcal{S}_e$ and $\mathcal{S}_c$ can be the total tasks set or if it is not possible for some tasks to complete within delay constraints, that will be considered as a failure of scheduling. Constraint Eq. 12 refers that the set $\mathcal{S}_e$ and $\mathcal{S}_c$ are both disjoint to each other.
After analyzing the problem, we have formulated, we observed that the problem is NP-complete according to the paper [14], although getting the optimal solution is time-consuming and the QoE of UEs will be drastically reduced if we try to get the optimal result.

4.2 Genetic Algorithm-based Tasks Assignment

The optimal solution is not feasible considering the task's delay deadline. In this regard, we designed a meta-heuristic approach to solve the problem. The genetic algorithm runs based on the generations of the population. The UEs are offloading tasks to the MEC server at the rate of λ. The MEC server will consider the newly arrived tasks and the remaining tasks in the previous time slot to make the total tasks set $\mathcal{S}_n$. The chromosome represents the combination of genes.
In Fig. 2 we represented chromosomes considering six tasks, where the index of the array represents the task number, and the value of each indexed cell represents the decision factor x_i to represent the single gene of each chromosome. The $x_i = 1$ indicates the task i will be placed in the $\mathcal{S}_e$ set and will be executed in the MEC server. And $x_i = 0$ means the tasks will be placed in S_c set and will be executed by IC. We made a population that is a reasonable subset from all possible task arrangements using simple random sampling without replacement approach. As the



(a) Crossover between two parent chromosomes



(b) Mutation of parent chromosome

Fig 2. Crossover and Mutation operation to generate new offspring in GATA.

size of $\mathcal{S}_n$ is n then a task $\mathcal{A}_i$ will be picked with probability $1/n$, which indicates all tasks are equally likely to be sampled. In Algorithm 1 we presented the proposed meta-heuristic approach using a genetic algorithm for task assignment in MEC and IC servers. After generating the initial population, we have to - calculate the fitness of each chromosome. Fitness can be calculated using Eq. 13

$$\mathcal{F} = \left(\sum_{i \in \mathcal{S}_e} T_e^i + \sum_{j \in \mathcal{S}_c} T_c^j \right) \times m. \tag{13}$$

The value of $\mathcal{F}$ includes the total delay for n number of tasks either executed on the MEC or on the IC server, multiplied by the weight factor. Here m in Eq. 13 is the weight factor and the initial value is 1 which indicates all the tasks are executed within the delay deadline of each task.

The value of m will increment by one for every task's failure (can not be executed within delay constraint). The intuition to multiply the weight factor is to increase the value of $\mathcal{F}$ for a particular chromosome having many failures. Hence when we sort chromosomes, then the worst chromosomes having a large value of $\mathcal{F}$ will be ranked lower. Now we will sort the chromosome according to the increasing fitness value which means the chromosome which has a combination of minimum delay requirements and lower failure of tasks and will be ranked higher.

After sorting the initial population in the increasing order of fitness value, we choose the best two chromosomes as the parent and crossover them to get new offspring. To do so, first, a random value smaller than n will be selected and then two-parent chromosome will interchange their genes from the selected index. As depicted in Fig. 2b, let's say the selected index is 5, then the genes from 5^{th} to the last index from parent 1 will be replaced by parent 2 genes and vice versa. For mutation, we have used the bit-flip mutation approach. In the mutation process, a random index or task number will be selected, and the value of that index will be flipped. For example, in Fig. 2b, the 4^{th} task is expected to be executed in IC, although by mutation we changed the bit so that the task is now expected to be executed in the MEC server. We considered the crossover and mutation probability according to [17]. This process will continue until the termination condition is met. Finally, the best chromosome based on the fitness value will be selected to make the task sets $\mathcal{S}_e$ and $\mathcal{S}_c$. Where i^{th} task will be placed in the MEC set or IC set considering the value of decision factor x_i and the selected server will complete the execution of the task.

Algorithm 1 Genetic Algorithm Based Task Assignment for Near-Optimal Task Scheduling

Input: set $\mathcal{S}_n$
Number of Iteration = I
Crossover probability = $\mathcal{P}_c$
Mutation probability = $\mathcal{P}_{\Updownarrow}$
Output: set $\mathcal{S}_e$, set $\mathcal{S}_c$

1. Make tasks population for $\mathcal{S}_n$ using simple random sample without replacement (SRSWOR)
2. for $i = 1$ to I do

(a) Calculate the fitness ($\mathcal{F}$) value of each chromosome using Eq. 13

(b) Sort population based on fitness value in increasing order

(c) if random$(0,1) < \mathcal{P}_c$ then

i. Select crossover index randomly

ii. Perform crossover between best two genes

(d) if random$(0,1) < \mathcal{P}_{\Updownarrow}$ then

i. Perform bit-flip mutation

(e) Add newly created offspring by crossover or mutation or by both to the next generation

3. Retrieve the first gene as the best possible solution of $\mathcal{S}_n$

4. Make $\mathcal{S}_e$ where value in the tasks index is 1

5. Make $\mathcal{S}_c$ where value in the tasks index is 0

Considering the number of tasks or number of genes in each chromosome N, the number of iterations I, population size M, the complexity of the proposed GATA algorithm is - $O(NM)$ for initial population generation, $O(N)$ for calculating fitness value, $O(N \log N)$ for sorting the chromosomes according to the ascending order of the fitness value, $O(N)$ for single-point crossover and mutation, and finally $O(N)$ for task sets generation. Hence total complexity will be $O(NM + I(N + N \log N + N) + N) \approx O(NM + IN \log N)$.

5 Performance Evaluation and Result Analysis

We have evaluated our algorithm based on the assumptions and constraints. We followed the similar system environment and setup requirement of Cloudsim and device configuration as described in [14]. We set the simulation environment considering image processing tasks and the task arrival rate is between 10 to 90 per second, having delay deadline distribution between 250 to 400 ms. Allocated bandwidth by MEC to UE is 10 Mbps, data size is 5MB, and tasks instruction size 600 ~ 1000 MIPS. For the MEC server and IC server, we followed the SpaceShared method in the CloudSim tool for CPU and memory allocation of the virtual machines (VMs). For the comparative study we compared our proposed GATA with three state-of-the-art works HTA [14], where authors proposed a heuristic algorithm based on priority factor calculated using MEC and IC execution time and CSS [13], where authors considered binary decision method to offload either in MEC or in IC. The GATA is also compared with CEGA [12], where authors only solved the offloading decision-making problem without considering queuing delay and tasks execution delay of MEC or IC server.

5.1 Performance Metrics

5.1.1 Success Rate of Tasks Execution:

The average number of tasks successfully executed within the constraints of the UEs is measured and the Percentage is calculated by this method. In this case, we considered all the tasks either executed on MEC or on IC. The tasks which delay deadlines were not satisfied are considered failures, but the execution will be done in any one of the servers.

5.1.2 Average Turnaround Time of Tasks:

This is calculated for the successful completion of tasks. When a task of any UE is successfully offloaded and executed and the server provides the resulting feedback, the time required for all these actions will be considered in this metric.

5.1.3 Average Service Provision Time:

Service Provision can be described as providing some services to some entity that require external computation or other resource-oriented services. A system will be considered to perform better if its service provision time is much smaller, that is, it can provide services faster to the incoming tasks. Faster service provision indicates better the QoE of UEs.

5.2 Results and Discussion:

We studied the performance of the GATA method by changing the task's arrival rate and changing the number of VMs in the MEC server.

5.2.1 Impact of Varying the Tasks Arrival Rate:

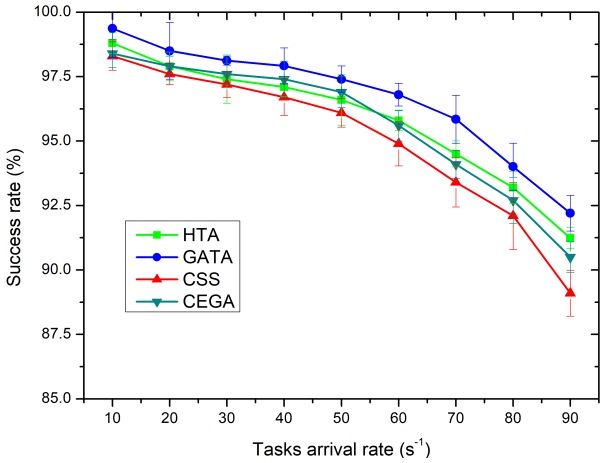
In Fig. 3 we observe the impact of varying task arrival rates at the MEC server. In this case, we varied the task arrival rate from 10 to 90 tasks per second. The number of VMs in the MEC server is considered 7. Task size and other parameters are generated using random distribution according to the mentioned range.
Fig. 3a represents the successful task execution rate for all the methods. We can see that when the task arrival rate is low like 10 or 20 tasks per second, HTA, CSS, CEGA, and GATA have a nearly similar success rate. When the arrival

rate increases, the success rate decreases for all the methods. The GATA outperforms the rest of the methods as in this approach we are using a meta-heuristic strategy and we run our algorithm for several hundreds of iterations. which ensures a near-optimal result for our formulated optimization problem. This also indicates that the GATA generated better task assignment decisions than other approaches, especially CEGA. As in CEGA, authors only considered task offloading scenarios. But in GATA we considered the user's tasks offloading and execution delay along with queuing delay in remote servers to make the decision for task completion.

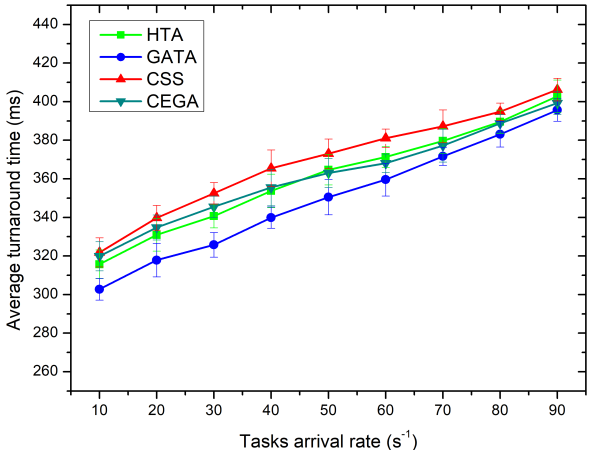
Hence, GATA returns the best possible server selection decisions for task execution. Therefore, this method improved the successful task completion rate compared to other approaches.

In Fig. 3b we can observe the average turnaround time (TAT) of tasks increases when the task arrival rate increases. When the incoming tasks increase at MEC then due to resource shortage MEC has to send more tasks to the IC server. Hence the turnaround time of tasks increases. Although the proposed GATA approach has achieved less TAT for the UEs and outperforms the heuristic approach HTA and binary decision-based approach CSS. Also, GATA ensures better TAT than CEGA since, the CEGA approach emphasize only on offloading latency. As MEC is closer to the UEs than IC, the CEGE method will try to choose MEC as the best possible server for offloading, which will increase task queuing in MEC. This will result in increasing TAT for tasks.

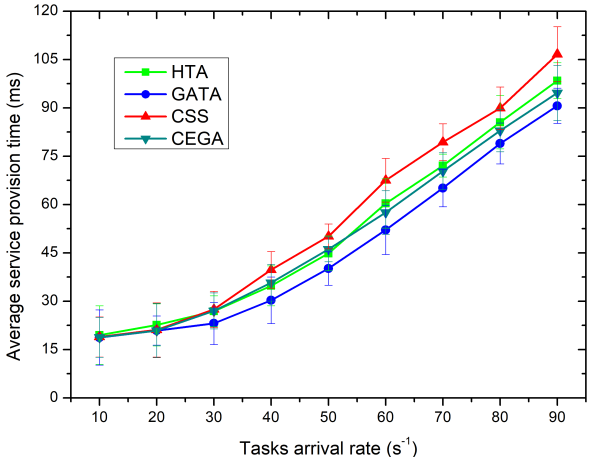
In Fig. 3c the average service provision time is measured for all methods. At the very beginning when the task arrival rate is very low then the service provision time for the tasks is very low and nearly similar for all the related works. When the arrival rate increases the service provision time also increases. As the increased number of tasks requires more resources to execute, the MEC cannot afford to provide resources due to resource shortage. In that case, more tasks are sent to the IC and the tasks need to wait more time to get the required service. The GATA ensures less waiting time for the tasks getting service, than other strategies. The reason behind this is, that in GATA when MEC observes a large number of tasks that need to be completed, then analyzing the load status of itself and IC, MEC will make better decisions for selecting the tasks execution server. Which is numerically depicted in the graph.



(a) Success Rate vs Arrival Rate λ



(b) Average Execution Time vs Arrival Rate λ

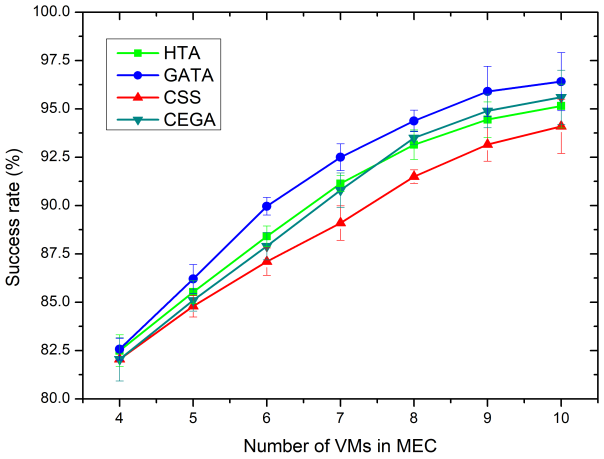


(c) Average Service Provision Time vs Arrival Rate λ

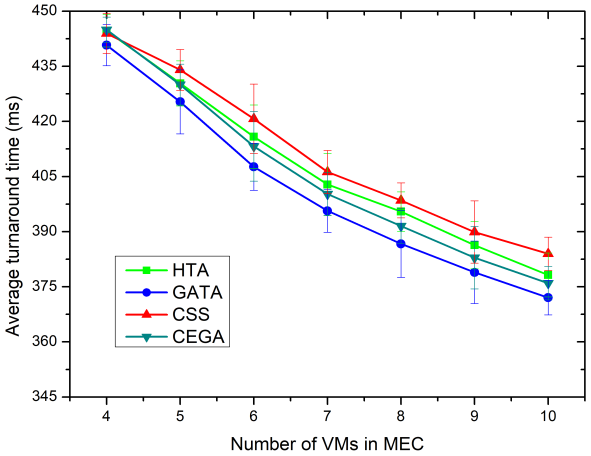
Fig 3. Impact for Different Arrival Rate of Tasks

5.2.2 Impact of Varying the VMs in MEC

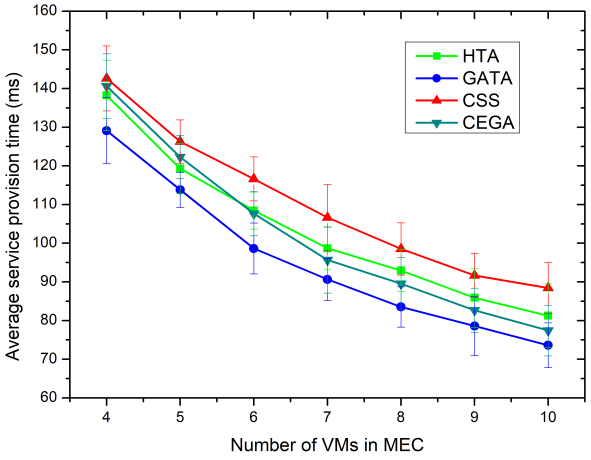
Fig. 4 represents the impact of varying the number of VMs in the MEC server. For this case, we considered the task's



(a) Success Rate vs Changing the Number of VMs



(b) Average Execution Time vs Changing the Number of VMs



(c) Average Service Provision Time vs Changing the Number of VMs

Fig 4. Impact of No. of VMs in MEC Server

arrival rate as 90 per second and changed the VM number in the MEC server.

Fig. 4a depicts the successful tasks execution rate of tasks. As we are increasing the VM number in MEC it means we are increasing computation resources at the edge. Hence when we are increasing the VMs more tasks can be easily served by the MEC server. Thus, the rate of successful task execution increases. The proposed GATA approach ensures more success rate compared to HTA, CSS, and CEGA.

In Fig. 4b we can see that the average turnaround time decreases for all the mentioned methods and the GATA has a very decent TAT than the rest of the approach. As the increased number of VMs indicates MEC resources are increasing, hence more tasks can be executed in the MEC server. In this case the queuing delay will be reduced, and more tasks can be executed parallelly.

Similarly in Fig. 4c, we observe that, the average service provision time decreases due to the increment of MEC resources in terms of VM. This ensures tasks will face less delay in getting service at MEC. The proposed GATA outperforms other works by taking decisions intelligently using genetic algorithm.

6 Conclusion

In this paper, we have described the task offloading and execution problems collaborating with MEC and IC servers. Tasks offloaded from UEs must be executed within the delay constraint, hence MEC has to make decisions intelligently considering task offloading delay and execution delay along with queuing delay at either in MEC or IC server. We proposed the GATA method for this scenario based on the genetic algorithm due to the complexity of the optimal decision-making problem of the MEC server. The GATA indicates a significant impact on successful task execution rate, reducing TAT and service provision latency. In the future, we will consider a multi-MEC scenario for resource sharing and interaction among them. We will explore different machine-learning techniques to solve the problem of resource sharing to improve edge resource utilization.

References

[1] M. Goudarzi, H. Wu, M. Palaniswami, and R. Buyya, "An application placement technique for concurrent iot applications in edge and fog computing environments," *IEEE Transactions on Mobile Computing*, vol. 20, no. 4, pp. 1298–1311, 2020.

[2] M. Shahzad, A. X. Liu, and A. Samuel, "Secure unlocking of mobile touch screen devices by simple gestures: You can see it but you can not do it," in *Proceedings of the 19th annual international conference on Mobile computing & networking*, 2013, pp. 39–50.

[3] S. Abolfazli, Z. Sanaei, and A. Gani, "Mobile cloud computing: A review on smartphone augmentation," *Information Systems, and Communications*, 2012.

[4] J. Ren, D. Zhang, S. He, Y. Zhang, and T. Li, "A survey on end-edge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet," *ACM Computing Surveys (CSUR)*, vol. 52, no. 6, pp. 1–36, 2019.

[5] F. Vhora and J. Gandhi, "A comprehensive survey on mobile edge computing: Challenges, tools, applications," in *2020 fourth international conference on computing methodologies and communication (ICCMC)*, IEEE, 2020, pp. 49–55.

[6] J. Ahmed, M. A. Razzaque, M. M. Rahman, S. A. Alqahtani, and M. M. Hassan, "A stackelberg game-based dynamic resource allocation in edge federated 5g network," *IEEE Access*, vol. 10, pp. 10 460–10 471, 2022.

[7] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM transactions on networking*, vol. 24, no. 5, pp. 2795–2808, 2015.

[8] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *2016 IEEE international symposium on information theory (ISIT)*, IEEE, 2016, pp. 1451–1455.

[9] E. Gelenbe, R. Lent, and M. Douratsos, "Choosing a local or remote cloud," in *2012 second symposium on network cloud computing and applications*, IEEE, 2012, pp. 25–30.

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

[11] S. Abdullah, E. Akter, M. S. Islam, and J. Ahmed, "Cost-effective iot-based smart stick for visually impaired person," in *2022 4th International Conference on Sustainable Technologies for Industry 4.0 (STI)*, IEEE, 2022, pp. 1–6.

[12] S. C. Sinthiya, N. I. Shuvo, R. R. Mahmud, and J. Ahmed, "Low-cost task offloading scheme for mobile edge cloud and internet cloud using genetic algorithm," in *2022 4th International Conference on Sustainable Technologies for Industry 4.0 (STI)*, IEEE, 2022, pp. 1–6.

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

[14] J. Ahmed, M. M. Ashhab, M. A. Razzaque, and M. M. Rahman, "Execution delay-aware task assignment in mobile edge cloud and internet cloud," in *2019 International Conference on Sustainable Technologies for Industry 4.0 (STI)*, IEEE, 2019, pp. 1–6.

[15] M. Bouet and V. Conan, "Mobile edge computing resources optimization: A geo-clustering approach," *IEEE Transactions on Network and Service Management*, vol. 15, no. 2, pp. 787–796, 2018.

[16] T. D. Novlan, H. S. Dhillon, and J. G. Andrews, "Analytical modeling of uplink cellular networks," *IEEE Transactions on Wireless Communications*, vol. 12, no. 6, pp. 2669–2679, 2013.

[17] D. E. Goldberg, *Genetic algorithms*. pearson education India, 2013.

Jargis Ahmed was born in Rajshahi, Bangladesh, in 1994. He received the Bachelor of Science, B.Sc. and Master of Science, M.Sc. in Computer Science and Engineering (CSE) from the University of Dhaka, Bangladesh, in 2017 and 2021 respectively. He has several publications in renowned conferences and journals. Currently, he is working as a Lecturer and Program Coordinator in the Department of CSE at Green University of Bangladesh (GUB) since 2018. His research interests include the Internet of Things (IoT), Mobile Edge Computing, Cloud Computing and 5G.