Joint Optimization of Task Offloading and Multicast Transmission Scheduling in MEC Networks

Ismail Bagayoko, Gael S. Mubibya, and Elmahdi Driouch
Université du Québec à Montréal, Montréal, QC, Canada
bagayoko.ismail@courrier.uqam.ca, mubibya.gael@courrier.uqam.ca, driouch.elmahdi@uqam.ca,

***Abstract*—The adoption of Multi-Access Edge Computing (MEC) stems from the need to overcome the limitations of mobile device resources and to optimize the performance of computation-intensive applications such as virtual and augmented reality. Additionally, by leveraging the content reuse property, multicast transmission emerges as an efficient strategy to alleviate network load, enabling the delivery of common content to multiple users simultaneously. This work investigates the challenges and opportunities associated with integrating multicast into MEC networks. We formulate a complex decision-making problem involving task offloading at the MEC level, designing multicast groups, and transmission scheduling. We develop a greedy heuristic solution along with a genetic algorithm-based approach. The simulation results provide valuable insights into the fundamental benefits of multicast in MEC networks, demonstrating its potential to optimize resource usage. They also highlight the superiority of our genetic approach as it efficiently balances multicast and unicast transmissions.**

***Index Terms*—Mobile edge computing, scheduling, resource allocation, task offloading, genetic algorithm, heuristic, multicast.**

I. Introduction

Mobile applications such as augmented/virtual reality (AR/VR), gesture recognition, and real-time gaming are highly demanding in terms of computation and latency. However, the limited processing power and energy constraints of user equipment (UE) hinder their ability to meet these requirements [1]. To address this, Multi-Access Edge Computing (MEC) has emerged as a promising paradigm, enabling the offloading of computational tasks to edge servers, thereby reducing latency and energy consumption [2]. Numerous studies have underscored the growing need for MEC. For instance, works in [3] and [4] emphasize the importance of intelligent and adaptive offloading strategies, particularly in dynamic and resource-constrained environments. According to [5], task offloading decisions are influenced by several factors, including required CPU cycles, input/output data sizes, energy budgets, deadlines, and channel conditions. These decisions can be made using static or dynamic policies, and under centralized or distributed control. To support efficient offloading, a range of algorithmic solutions have been proposed. Optimization-based methods such as binary integer programming [6] and Markov Decision Processes (MDPs) [7] aim to maximize system utility by jointly optimizing resource allocation and scheduling. However, due to the computational complexity of these techniques in large-scale networks, metaheuristic approaches have gained popularity. For example, authors in [8] developed an improved genetic algorithm to reduce task delay, while [9] proposed a genetic algorithm that maintains population diversity under dynamic conditions. Other studies [10], [11] have combined genetic algorithms with conflict graphs or swarm intelligence to enhance scalability and robustness.

Most existing works on task offloading and resource allocation in MEC systems focus only on unicast transmissions. However, multicast can offer substantial benefits, particularly for high-bandwidth applications [12]. Work in [13] proposed a multicast-based resource allocation and caching strategy in heterogeneous MEC networks. While effective in reducing delay and energy consumption, multicast usage is assumed rather than optimized. [14] analyzed MEC capacity under dynamic user sessions and content popularity, deriving theoretical capacity bounds. However, task offloading was not addressed. [15] introduced a multicast-aware cooperative caching strategy to reduce latency, though explicit multicast scheduling was not the focus. [16] examined multicast-based task offloading in vehicular networks, using interior-point optimization to minimize delay, showing clear latency improvements over baselines. [17] jointly optimized task offloading and multicast under caching, latency, and energy constraints, using a low-complexity algorithm based on the concave-convex procedure and ADMM. Unlike [17], our work considers both input and output transmissions during task offloading, without assuming that MEC servers function only as caches. This is critical for VR-type applications, where output data size and delivery timing significantly impact performance.

Some works have explored the integration of multicast with MEC and caching. By combining multicast transmission with edge caching, frequently accessed data can be reused across users, thereby alleviating network congestion. Studies such as [17], [18] demonstrate how this synergy improves delivery efficiency and reduces redundant data transfers.

While MEC and multicast each provide substantial performance gains individually, jointly leveraging them poses new challenges. Coordinating task offloading with dynamic multicast introduces complex scheduling and resource allocation problems, especially under latency and bandwidth constraints.

To address these challenges, this work proposes a joint optimization framework that integrates task offloading, multicast group design, and transmission scheduling. We formulate the problem as a Binary Integer Linear Program (BILP) and develop two algorithmic solutions: a greedy heuristic and a genetic algorithm. We show that incorporating multicast

979-8-3503-9281-4/25/$31.00 ©2025 IEEE

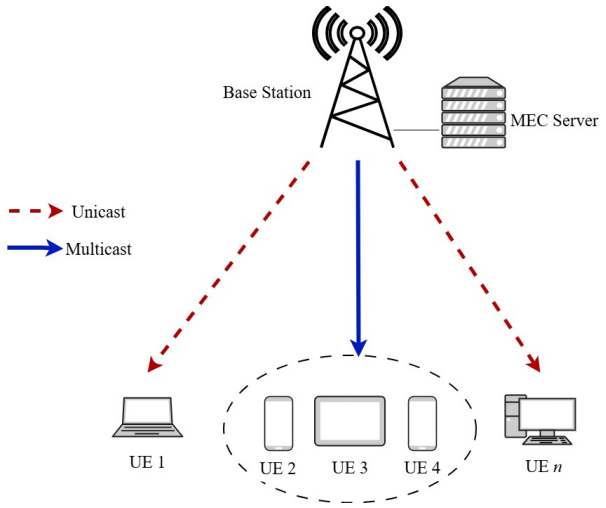

Fig. 1: System model

into MEC significantly enhances resource utilization and task completion rates, particularly in high-demand scenarios.

The main contributions of this work are as follows:

- We formulate a joint task offloading and multicast transmission scheduling problem in MEC networks, incorporating caching and latency constraints.
- We propose two solution strategies: a low-complexity greedy heuristic and a genetic algorithm designed to explore a wider range of possible scheduling configurations.
- We evaluate both methods through simulations, demonstrating performance gains of multicast-aware offloading.

The remainder of the paper is organized as follows: section II presents the system model and problem formulation. Section III describes the proposed algorithms. Section IV analyzes the simulation results. Section V concludes and outlines future directions.

II. System Model and Problem Formulation

A. System Model

As illustrated in fig. 1, we consider a MEC system composed of a single base station (BS) with direct access to a MEC server, and a set $\mathcal{N} = \{1, \dots, n, \dots, N\}$ of UEs. We adopt a discrete-time model, where time is divided into equal intervals of length T_s and indexed by $\mathcal{T} = \{0, \dots, t, \dots T\}$. We assume that the BS is able to predict the upcoming task requests, enabling us to explore performance upper bounds.

1) Request and Task Model: Each task $k \in \mathcal{K}$ is characterized by a computation load $\omega_k = \psi_k / I_i$, where ψ_k (*in cycles/bit*) is the resource need of task k and I_i (*in bits*) is the size of input data $i \in \mathcal{I}$. Additionally, each task has an output ratio $\beta_k = O_{k,i}/I_i$, where $O_{k,i}$ denotes the size of the output generated from input i by task k. Each request $\alpha \in \mathcal{A}(t)$ is defined by a tuple $(n_\alpha, \kappa_\alpha, \iota_\alpha, \tau_\alpha, \delta_\alpha)$, where n_α is the index of the UE generating the request, κ_α is the task index, ι_α is the input data index, τ_α is the arrival time slot, and δ_α is the deadline slot by which the task must be completed.

Finally, let $\mathcal{K} = \{1, \dots, K\}$ denotes the set of tasks, and $\mathcal{I} = \{1, \dots, I\}$ the set of input data cached at the MEC server.

2) Caching and Communication Model: The MEC cache may store both input and output data. Let $\mathbf{Z} \triangleq [Z_i(t)]$ and $\mathbf{Z}^{\text{out}} \triangleq [Z^{\text{out}}_{k,i}(t)]$ denote the matrices of binary caching indicators. Specifically,

$$Z_i(t) = \begin{cases} 1, & \text{if input data } i \text{ is cached at time slot } t, \\ 0, & \text{otherwise} \end{cases}$$

and

$$Z^{\text{out}}_{k,i}(t) = \begin{cases} 1, & \text{if the output of task } k \text{ executed on input } i \\ & \text{is cached at time slot } t, \\ 0, & \text{otherwise.} \end{cases}$$

The communication process begins with the BS broadcasting a beacon frame to all UEs. This allows each UE to verify whether it has pending computation requests. Then, each UE responds via uplink by sending the necessary information to trigger the execution of a task on specific input data. The MEC server scheduler then decides whether the task should be executed locally at the UE or offloaded to the MEC server. Finally, the BS communicates the scheduling decisions back to the relevant UEs, including multicast group assignments and task execution schedules.

Following these steps, the BS chooses to communicate with UEs either individually via unicast or collectively via multicast. In the unicast mode, each UE n is served with an achievable data rate given by:

$$R_n(t) = B \cdot \log_2 \left(1 + \frac{P \cdot h_n(t)}{N_0}\right), \tag{1}$$

where B is the available bandwidth, P is the BS transmission power, N_0 is the total noise power on the channel, and $h_n(t)$ is the channel gain between the BS and UE n at time slot t. Alternatively, the BS can group UEs requesting the same input or output data and perform a multicast transmission. The corresponding multicast rate is defined as:

$$R_{k,i}(t) = \min_{n \in \mathcal{G}} R_n(t), \tag{2}$$

where $\mathcal{G} \subseteq \mathcal{N}$ denotes the set of UEs in the multicast group.

At each time slot t, the BS schedules multicast transmissions based on the execution mode of each request. It either transmits input data i for local execution or sends the output of a task processed at the server. To capture these decisions, we define matrices $\mathbf{X} \triangleq [X_{n,i}(t)]$ and $\mathbf{Y} \triangleq [Y_{n,k,i}(t)]$, where

$$X_{n,i}(t) = \begin{cases} 1, & \text{if input } i \text{ is sent to UE } n \text{ at time slot } t, \\ 0, & \text{otherwise} \end{cases}$$

and

$$Y_{n,k,i}(t) = \begin{cases} 1, & \text{if the output of task } k \text{ on input } i \text{ is} \\ & \text{sent to UE } n \text{ at time slot } t, \\ 0, & \text{otherwise.} \end{cases}$$

Note that, at each time slot t, the uplink transmission delay is considered negligible.

3) Computation Model: Each request α can be executed either locally at the UE or remotely at the MEC server. In the local execution mode, the input data ι_α is transmitted to UE n_α, with a transmission delay of:

$$\Delta_\alpha^{\text{in}} = \frac{I_{\iota_\alpha}}{R_{\kappa_\alpha,\iota_\alpha}}, \tag{3}$$

followed by local processing time of:

$$\Delta_\alpha^{\text{exec}} = \frac{I_{\iota_\alpha} \cdot \omega_{\kappa_\alpha}}{f_{n_\alpha}}, \tag{4}$$

where f_{n_α} is the computational capacity of UE n_α. The total time for local execution is then:

$$\Delta_\alpha^{\text{local}} = \Delta_\alpha^{\text{in}} + \Delta_\alpha^{\text{exec}}. \tag{5}$$

If the task is executed at the MEC server, the computation delay is:

$$\Delta_\alpha^{\text{exec}} = \frac{(1 - Z^{\text{out}}_{\kappa_\alpha,\iota_\alpha}(t)) \cdot I_{\iota_\alpha} \cdot \omega_{\kappa_\alpha}}{F}, \tag{6}$$

where F is the CPU frequency of the MEC server. The output is then downloaded by the UE with a delay:

$$\Delta_\alpha^{\text{out}} = \frac{I_{\iota_\alpha} \cdot \beta_{\kappa_\alpha}}{R_{\kappa_\alpha,\iota_\alpha}}. \tag{7}$$

Thus, the total delay for MEC execution is:

$$\Delta_\alpha^{\text{MEC}} = \Delta_\alpha^{\text{exec}} + \Delta_\alpha^{\text{out}}. \tag{8}$$

The overall execution time for request α is given by:

$$\Delta_\alpha = X_{n,i}(t) \cdot \Delta_\alpha^{\text{local}} + Y_{n,k,i}(t) \cdot \Delta_\alpha^{\text{MEC}}. \tag{9}$$

B. Problem Formulation

The objective of this work is to maximize the number of requests completed within their latency deadlines while respecting the cache capacity of the MEC server. To formulate this, we define the binary matrix $\mathbf{U} \triangleq [U_\alpha]$, where

$$U_\alpha = \begin{cases} 1, & \text{if } \Delta_\alpha \le \delta_\alpha, \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

Here, $U_\alpha = 1$ indicates that the request α satisfies its deadline constraint. The optimization problem can then be formulated as follows:

$$\underset{\mathbf{X},\mathbf{Y},\mathbf{Z},\mathbf{Z}^{\text{out}},\mathbf{U}}{\text{Maximize}} \quad \sum_{\alpha \in \mathcal{A}(t)} U_\alpha \tag{11}$$

Subject to:

$$\sum_{i \in \mathcal{I}} X_{n,i}(t) \le 1, \quad \forall t \in \mathcal{T}, \forall n \in \mathcal{N} \tag{12}$$

$$\sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{I}} Y_{n,k,i}(t) \le 1, \quad \forall t \in \mathcal{T}, \forall n \in \mathcal{N} \tag{13}$$

$$X_{n,i}(t) + Y_{n,k,i}(t) \le 1, \quad \forall t \in \mathcal{T}, \forall n \in \mathcal{N}, \forall i \in \mathcal{I}, \forall k \in \mathcal{K} \tag{14}$$

$$\sum_i Z_i(t) I_i + \sum_{k,i} Z^{\text{out}}_{k,i}(t) O_{k,i} \le D, \quad \forall t \in \mathcal{T} \tag{15}$$

$$X_{n,i}(t), Y_{n,k,i}(t), Z_i(t), Z^{\text{out}}_{k,i}(t), U_\alpha \in \{0,1\} \tag{16}$$

Constraints (12) ensure that at most one input data transmission occurs per UE per time slot. Constraints (13) guarantee that at most one output data transmission is allowed per UE per time slot. Constraints (14) prevent simultaneous input and output transmissions for the same UE. Constraints (15) ensure that the total size of cached input and output data does not exceed the MEC server's cache capacity D. Finally, constraints (16) enforce the binary nature of the decision variables $X_{n,i}(t)$, $Y_{n,k,i}(t)$, and U_α.

Our problem shares significant similarities with the classical Job Shop Scheduling with Deadlines. The decision variables, such as X and Y, along with the formulated constraints, reflect characteristics commonly found in scheduling problems. This analogy highlights the inherent complexity of our formulation, particularly due to the strict temporal constraints. Since scheduling problems are well known to be NP-hard, the structural resemblance suggests that our problem is also computationally challenging. Therefore, we assume that the caching decisions $\mathbf{Z}$ are predefined and focus primarily on offloading and scheduling decisions.

III. Proposed Solutions

A. Greedy Heuristic Algorithm (GHA)

1) Algorithm Description: Due to the computational complexity of the problem, we propose a greedy heuristic algorithm (GHA) that prioritizes low execution time over optimality. GHA selects requests with the earliest deadlines and schedules them in the earliest available time slots.

As shown in Algorithm 1, at each iteration, GHA performs the following steps:

- **a.** *Request Selection*: Select the request with the closest deadline.
- **b.** *Grouping Options*: Form two candidate groups based on the selected request: one for local execution (input transmission) and one for MEC execution (output transmission).
- **c.** *Slot Assignment*: For each group, assign the earliest conflict-free time slot. If no valid slot exists, the request is dropped.
- **d.** *Planning*: Select the group that satisfies the most requests within deadline constraints and schedule it. Remove scheduled requests from the pending set.
- **e.** *Repeat*: Repeat the above steps until no more requests can be scheduled.

This deadline-driven approach efficiently reduces waiting time and ensures timely execution.

2) Complexity Analysis: In the worst case, where no grouping is possible, GHA must evaluate each request individually for every time slot. The outer loop runs up to $|\mathcal{A}|$ times, and each grouping check may involve all remaining requests and time slots. Thus, the worst-case complexity is $\mathcal{O}(|\mathcal{A}|^2 + |\mathcal{A}| \cdot T)$, where $|\mathcal{A}|$ is the number of requests.

Algorithm 1: Greedy Heuristic Algorithm (GHA)

Input: Set of pending requests $\mathcal{A}$, set of time slots $\mathcal{T}$
Output: Scheduled requests and their transmission slots
while $\mathcal{A} \neq \emptyset$ **do**
  Select request α with the earliest deadline from $\mathcal{A}$
  Form group $\mathcal{G}_{\text{input}}$ for local execution based on α
  Form group $\mathcal{G}_{\text{output}}$ for MEC execution based on α
  Find earliest conflict-free time slot t_{in} for $\mathcal{G}_{\text{input}}$
  Find earliest conflict-free time slot t_{out} for $\mathcal{G}_{\text{output}}$
  if $t_{in} = \varnothing$ *and* $t_{out} = \varnothing$ **then**
    Remove α from $\mathcal{A}$
  else
    Choose $\mathcal{G}^*$ as the group satisfying the most requests
    Assign earliest valid time slot t^* for $\mathcal{G}^*$
    Schedule $\mathcal{G}^*$ at time slot t^*
    Remove scheduled requests in $\mathcal{G}^*$ from $\mathcal{A}$

B. Genetic Algorithm for Request Scheduling (GARS)

1) Overview of Genetic Algorithms: Genetic Algorithms are population-based optimization methods inspired by natural selection [19]. To address the joint scheduling and offloading problem under latency and caching constraints, we propose a genetic algorithm-based strategy called Genetic Algorithm for Request Scheduling (GARS). GARS introduces problem-specific encoding and fitness functions tailored to the characteristics of MEC-aware multicast scheduling. Here are the steps:

- *Chromosome (Individual) Representation:* Each chromosome encodes a complete scheduling plan for all requests α as a two-row matrix. Each column corresponds to a single request. The first row indicates the execution decision:

$$\begin{cases} 1, & \text{if the task is executed locally,} \\ 2, & \text{if the task is executed at the MEC server,} \\ 0, & \text{if the request is rejected.} \end{cases}$$

  The second row specifies the transmission time slot assigned to the selected execution mode. This encoding allows GARS to jointly optimize offloading decisions and scheduling under deadline constraints.
- *Population Initialization:* A population of individuals $\mathcal{P} = \{I_1, I_2, \ldots, I_P\}$ is randomly initialized, where P denotes the population size. This randomization ensures a diverse set of scheduling configurations, which is critical for exploring trade-offs between deadline satisfaction and resource utilization.
- *Fitness Evaluation:* Each individual is evaluated using the Proportional Delay Score (S_{PD}), which rewards timely scheduling and penalizes late executions based on the severity of the delay:

$$S_{PD} = \sum_{\alpha \in \mathcal{A}} \begin{cases} 2, & \text{if } \text{delay}_\alpha \geq 4 \\ 1, & \text{if } 0 < \text{delay}_\alpha < 4 \\ 0, & \text{otherwise.} \end{cases} \quad (17)$$

  The delay for each request α is computed as:

$$\text{delay}_\alpha = \max\left(0, \Delta_\alpha + t_{\text{sched}}(\alpha) - (\tau_\alpha + \delta_\alpha)\right) \quad (18)$$

  Here, $t_{\text{sched}}(\alpha)$ denotes the transmission time slot assigned to request α. The score S_{PD} allows more flexibility than a hard deadline criterion by giving partial credit for moderately delayed requests, while still penalizing significant violations.
- *Genetic Operators:* Individuals are selected for reproduction using elitist selection, which preserves the best-performing candidates across generations. For crossover, we employ two-point and k-point strategies, where two or more crossover points are randomly chosen with a probability of p_c, and genetic material is exchanged between parents based on the resulting segments. To maintain diversity and avoid premature convergence, mutation is applied to each gene with a fixed probability p_m.
- *Survivor Selection, Correction, and Termination:* A new population is formed by selecting the fittest individuals from both parents and offspring. Optionally, a correction strategy may be applied after genetic operations to enforce feasibility constraints The algorithm terminates either after a predefined number of generations or when no significant improvement in population fitness is observed over consecutive iterations.

GARS provides a structured and adaptive approach to manage multicast-aware offloading decisions. It balances exploration and exploitation, supports deadline-driven optimization, and is particularly effective in dynamic environments where exhaustive search is impractical. **Algorithm** 2 summarizes the key steps of the proposed method.

2) Complexity Analysis: The computational complexity of the GARS algorithm is primarily determined by the number of generations $G_{\max}$, the population size P, and the number of requests $|\mathcal{A}|$ to be scheduled.

In each generation, elitist selection requires sorting the population based on fitness values, which incurs a complexity of $\mathcal{O}(P \log P)$ in the worst case. Both crossover and mutation operations are assumed to operate in $\mathcal{O}(|\mathcal{A}|)$ time per individual, and evaluating the fitness function also takes $\mathcal{O}(|\mathcal{A}|)$ per individual. Therefore, the total cost per generation is $\mathcal{O}(P \cdot |\mathcal{A}| + P \log P)$. Since the algorithm runs for $G_{\max}$ generations, the total time complexity becomes $\mathcal{O}\left(G_{\max} \cdot (P \cdot |\mathcal{A}| + P \log P)\right)$.

This complexity is linear with respect to the number of requests and generations, making GARS scalable for moderate-size scheduling problems. Furthermore, due to the inherent parallelism of genetic algorithms, significant acceleration can be achieved through parallel or distributed computing environments.

Algorithm 2: GARS

Input: $\mathcal{A}$, $G_{\max}$, $\mathcal{P}$, $p_m \in [0,1]$, $p_c \in [0,1]$
Output: Best individual (schedule)

Initialize population $\mathcal{P}$ with random chromosomes
for $g \in G_{\max}$ **do**
  Evaluate fitness of $I_i \in \mathcal{P}, \forall i \in \mathcal{I}$, using Eq. (17)
  Select parents from $\mathcal{P}$ based on elitist selection
  Apply crossover to generate offspring from selected parents with probability p_c
  Apply mutation to offspring with probability p_m
  Evaluate fitness of offspring
  Form new population $\mathcal{P}$ by selecting elitists from current population and offspring
Return the best individual in $\mathcal{P}$

IV. Performance Evaluation

A. Simulation Parameters

TABLE I: Notation and values of general parameters

Notation	Description	Value
N	Number of UEs	[10, 100]
K	Number of tasks	5
I_i	Input data size of task i	[1 Kb, 1 Mb]
$\lvert\mathcal{I}\rvert$	number of input data	5
β_k	Size output/input ratio	[1.5, 2]
ψ_k	Required CPU cycles for task k	[100 MHz, 1 GHz]
f_n	CPU frequency of UE n	100 MHz
F	CPU frequency of the MEC server	5 GHz
B	Channel bandwidth	20 MHz
P	Transmission power	23 dBm
N_0	Total noise power	-107 dBm
T_s	Time slot duration	1 second

TABLE II: Selected Parameters for GARS

Parameter / Operator	Value
Number of Generations $G_{\max}$	150
Population Size P	100
Repair Strategy	No correction
Mutation rate p_m	0.1
Crossover rate p_c	0.7

The simulation environment consists of a network with a single BS and a variable number of UEs. The UE positions are randomly generated according to a uniform distribution within a circular area with a radius of 500 meters. The BS is located at the center of this area. Other environment parameters are in table I. The parameters used for GARS is summarized in table II. Each simulation run spans at least 40 instances. At the beginning of each slot, each UE independently generates a task request with a probability of 0.05.

B. Simulation Results

Figure 2 illustrates the request execution rate as a function of the number of UEs. We compare the performance of GARS and GHA against a baseline algorithm that relies solely on unicast transmissions. This baseline prioritizes the most urgent requests. Each request is served individually: if local execution

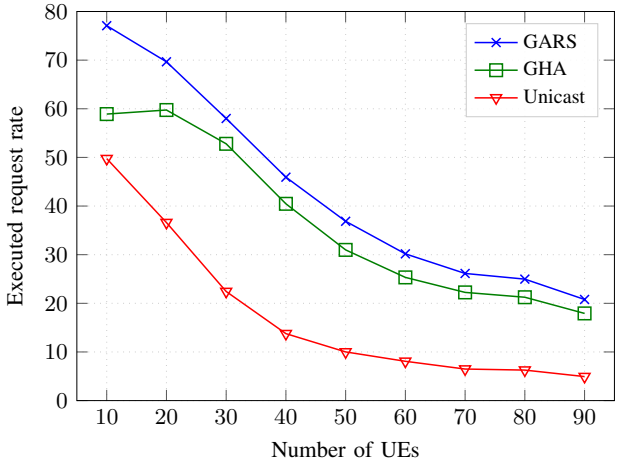

Fig. 2: Request execution rate vs. number of UEs

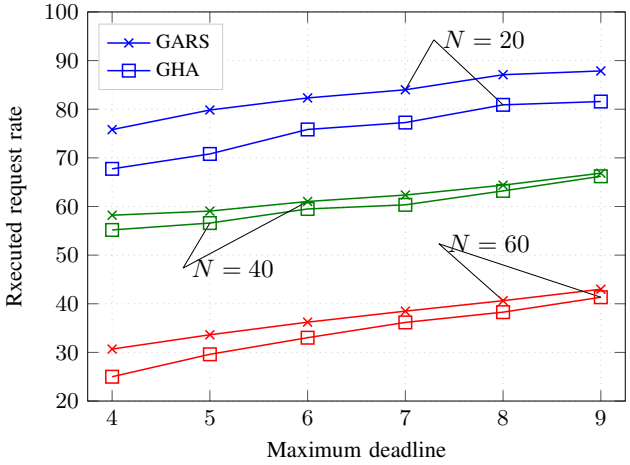

Fig. 3: Request execution rate vs. maximum deadline.

is feasible within the deadline, the input is sent to the user; otherwise, the task is offloaded to the MEC server, and the output is delivered. Since the request generation probability remains constant, a higher number of UEs leads to increased demand, causing the execution rate to decline due to limited system resources. Despite this general decline in execution rate with increasing UEs density, GARS consistently outperforms both GHA and the unicast baseline across all scenarios. This performance advantage stems from GARS's more effective use of multicast transmissions compared to GHA.

Figure 3 illustrates the impact of executed request rate when the maximum deadline changes for two different values of N. As the deadline increases, the performance of all algorithms improves. Longer deadlines provide the system with greater flexibility in scheduling, enabling more requests to be executed within the time constraints. This also facilitates a higher usage of multicast transmissions. When the deadlines are sufficiently large, the system can execute all requests, predominantly through multicast.

Figures 4 and 5 show the executed request rate as a

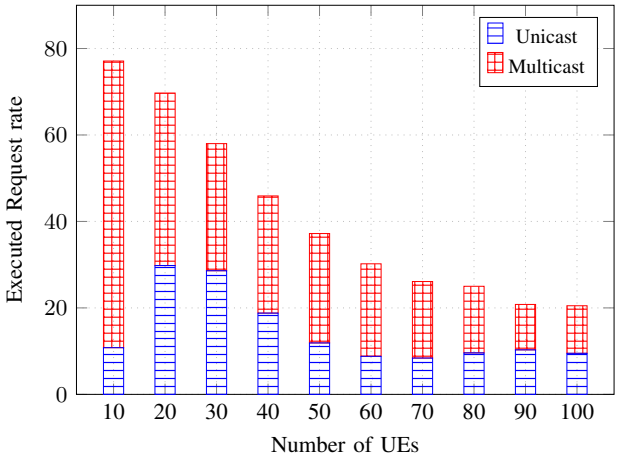

Fig. 4: GARS: Executed request rate vs. the number of UEs

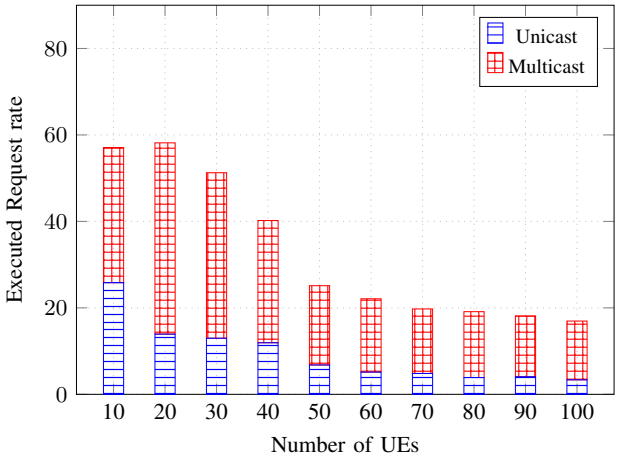

Fig. 5: GHA: Executed request rate vs. the number of UEs.

function of the number of UEs, along with the distribution between multicast and unicast transmissions for both GARS and GHA, respectively. GARS consistently outperforms GHA, not only due to a higher number of multicast transmissions, but also through more effective unicast scheduling. The multicast transmissions in GARS are more intelligently formed, often involving larger user groups than those of GHA. As a result, GARS is able to execute more requests overall, even in scenarios where GHA performs more multicast transmissions. This indicates that GARS achieves a better balance between multicast and unicast strategies, while GHA tends to prioritize multicast execution, sometimes at the expense of overall efficiency.

V. Conclusion

This paper addressed the joint task offloading and multicast transmission problem in MEC networks. We proposed two scheduling strategies: a greedy heuristic and a GARS. The greedy method provides low-complexity scheduling, while GARS leverages evolutionary search to improve request satisfaction under deadline and cache constraints. Simulation results showed that GARS outperforms both the heuristic and baseline unicast schemes, demonstrating the benefit of combining multicast and adaptive scheduling. Future work will explore dynamic output caching, inter-server coordination, user mobility, and the formulation of an online version of the scheduling problem where task requests and channel conditions are unknown in advance to the BS/MEC sever and evolve in real time.

References

[1] P. Mach and Z. Becvar, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading," *IEEE Commun. Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[2] T. Alfakih, M. M. Hassan, A. Gumaei, C. Savaglio, and G. Fortino, "Task Offloading and Resource Allocation for Mobile Edge Computing by Deep Reinforcement Learning Based on SARSA," *IEEE Access*, vol. 8, pp. 54 074–54 084, 2020.

[3] K. Cao, Y. Liu, G. Meng, and Q. Sun, "An Overview on Edge Computing Research," *IEEE Access*, vol. 8, pp. 85 714–85 728, 2020.

[4] L. Kong, S. Zhang, J. Cao, and D. Chen, "Edge Computing: A Survey on Technologies, Applications and Research Directions," *IEEE Access*, vol. 10, pp. 88 872–88 920, 2022.

[5] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A Survey on Mobile Edge Computing: The Communication Perspective," *IEEE Commun. Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[6] Y. He, C. Wang, L. Zhang, and Q. Liu, "Joint Task Offloading and Resource Allocation for Mobile Edge Computing Using Genetic Algorithm," *IEEE Access*, vol. 9, pp. 28 163–28 173, 2021.

[7] G. Yang, L. Hou, X. He, D. He, S. Chan, and M. Guizani, "Offloading Time Optimization via Markov Decision Process in Mobile-Edge Computing," *IEEE Internet of Things J.*, vol. 8, no. 4, pp. 2483–2493, 2020.

[8] A. Zhu and Y. Wen, "Computing Offloading Strategy Using Improved Genetic Algorithm in Mobile Edge Computing System," *Journal of Grid Computing*, vol. 19, no. 3, p. 38, 2021.

[9] M. S. Zalat, S. M. Darwish, and M. M. Madbouly, "An Adaptive Offloading Mechanism for Mobile Cloud Computing: A Niching Genetic Algorithm Perspective," *IEEE Access*, vol. 10, pp. 76 752–76 765, 2022.

[10] A. A. Al-Habob, O. A. Dobre, A. G. Armada, and S. Muhaidat, "Task Scheduling for Mobile Edge Computing Using Genetic Algorithm and Conflict Graphs," *IEEE Trans. on Veh. Technol.*, vol. 69, no. 8, pp. 8805–8819, 2020.

[11] A. Abbas, A. Raza, F. Aadil, and M. Maqsood, "Meta-Heuristic-Based Offloading Task Optimization in Mobile Edge Computing," *Intern. Journal of Distrib. Sensor Networks*, vol. 17, no. 6, 2021.

[12] O. Chukhno, K. Samouylov, and Y. Koucheryavy, "Multicast Commun. in 5G: State of the Art and Research Challenges," *IEEE Access*, vol. 11, pp. 36 928–36 949, 2023.

[13] H. Hao, C. Xu, S. Yang, L. Zhong, and G.-M. Muntean, "Multicast-aware optimization for resource allocation with edge computing and caching," *Journal of Network and Computer Applications*, vol. 193, 2021.

[14] P. Yuan, M. Li, S. Li, C. Liu, and X. Zhao, "Maximizing the Capacity of Edge Networks with Multicasting," *Applied Sciences*, vol. 13, no. 14, 2023.

[15] X. Huang, Z. Zhao, and H. Zhang, "Cooperate Caching with Multicast for Mobile Edge Computing in 5G Networks," in *in proc. IEEE VTC Spring*, 2017.

[16] H. Li, X. Li, M. Zhang, and B. Ulziinyam, "Multicast-Oriented Task Offloading for Vehicle Edge Computing," *IEEE Access*, vol. 8, pp. 187 373–187 383, 2020.

[17] Y. Sun, Z. Chen, M. Tao, and H. Liu, "Bandwidth Gain from Mobile Edge Computing and Caching in Wireless Multicast Systems," *IEEE Trans. on Wireless Commun.*, vol. 19, no. 6, pp. 3992–4007, 2020.

[18] Y. Zhou, F. R. Yu, J. Chen, and Y. Kuo, "Cache-Aware Multicast Beamforming Design for Multicell Multigroup Multicast," *IEEE Trans. on Veh. Technol*, vol. 67, no. 12, pp. 11 681–11 693, 2018.

[19] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: past, present, and future," *Multimedia tools and applications*, vol. 80, pp. 8091–8126, 2021.