Efficient Placement of Interdependent Services in Multi-access Edge Computing

Shuyi Chen[1,2][0000−0003−0745−4083], Panagiotis Oikonomou[3][0000−0002−5564−2591], Zhengchang Hua[1,2][0000−0002−3970−6129], Nikos Tziritas[3][0000−0002−2091−2037], Karim Djemame[2][0000−0001−5811−5263], Nan Zhang[1][0000−0002−5728−0440], and Georgios Theodoropoulos[1][0000−0002−7448−5886]⋆

[1] Southern University of Science and Technology, Shenzhen, China
[2] University of Leeds, Leeds, UK
[3] University of Thessaly, Lamia, Greece

Abstract. The rise of 5G fuels multi-access edge computing (MEC), a transformative computing paradigm that leverages edge resources for low-latency mobile access and complex service execution. Deploying services across geographically distributed edge nodes challenges providers to optimize performance metrics like latency and resource efficiency, impacting user experience, operational cost, and environmental footprint. In the context of service scheduling with data flow dependencies, we propose heuristic-based service placement algorithms that balance minimizing latency and maximizing resource efficiency. Our algorithms, evaluated in a simulated environment using state-of-the-art workload benchmarks, achieve significant energy consumption improvements while maintaining comparable latency.

Keywords: Service placement · Multi-access edge computing · Task dependency graph.

1 Introduction

Driven by the advent of fifth generation (5G) network, multi-access edge computing (MEC) emerges as a transformative paradigm by leveraging network and computing resources at the network edge. This proximity offers low-latency access for mobile subscribers and facilitates the execution of complex, computationally intensive applications [9]. However, effectively managing custom applications in MEC requires strategic task offloading. This involves strategically assigning tasks to the most suitable computing nodes, balancing the demands of both users (low latency) and providers (resource efficiency). Jobs submitted by users may surround IoT data processing, healthcare, Augmented Reality(AR)-based experiences or financial services. The offloading process considers the application's underlying requirements, and allocates sufficient resources at the edge to ensure their successful execution.

⋆ Corresponding author

Quality of Service (QoS), a measure of service effectiveness, is paramount for service providers in MEC. Delivering optimal QoS involves meeting multiple objectives such as latency, bandwidth, and availability. Providers achieve this by meticulously coordinating network and computing resources, down to individual task allocation [10]. Beyond QoS, profitability remains a key concern. Minimising server rental costs and power consumption directly impact profits and contribute to a more sustainable industry. However, achieving these goals often involves trade-offs [15]. Striking a balance between user satisfaction (low latency) and provider needs (cost efficiency) presents a multi-objective optimization problem. While existing research offers solutions for various scenarios, they may lack the necessary universality and flexibility required for the dynamic nature of MEC environments and complex applications.

Beyond managing the distributed nature of MEC resources, the placement of services presents an additional challenge. In real-world applications, numerous interdependent components frequently collaborate [16]. Each component executes a specific task such as data extraction, transformation, loading, or integration. The optimal placement of these components has to fulfil the requirements of each component with dependency guarantees.

Existing service placement strategies in edge computing struggle to effectively handle these complex, dependency-aware applications. Traditional methods often overlook the inter-dependencies of application modules, or leverage a specific architecture or model. To address this gap, we propose service placement approaches specifically designed for Cloud-MEC environments, aiming at efficiently allocate resources for complex service execution and fulfill both user and provider demands. Offloading occurs at edge nodes located close to user devices, where computational tasks are transferred to reduce latency. Computation-intensive tasks may also be offloaded to powerful cloud data centers.

This work addresses the service placement problem with precedence constraints among service applications. Our objective is to improve quality of service (QoS), specifically minimizing latency, and optimize resource efficiency, measured by energy consumption. The inherent complexity of considering these multiple factors motivates our development of two novel heuristic-based placement algorithms. These algorithms strive to achieve a balance between minimizing user-experienced latency and energy consumption. Our contributions in this work are twofold:

- Dependency-aware Service Placement Algorithms: We propose two novel service placement algorithms specifically designed for the multi-access edge computing (MEC) environment. These algorithms consider precedence constraints between service components to optimize both end-to-end latency and dynamic energy consumption.
- YAFS Platform Extension: To support the evaluation of our algorithms, we developed an extension [4] to the YAFS simulation platform [6]. This extension enables the modeling of sequential task processing, a crucial aspect of service execution in MEC.

[4] `https://github.com/Sukiiichan/YAFS_MEC`

The remainder of this paper is organized as follows. Section 2 briefly summarizes existing approaches and identifies the research gap. Section 3 describes the system models and problem formulation. Section 4 proposes our two algorithms in detail. Simulation results are presented in Section 5, and the paper concludes in Section 6.

2 Related work

The multi-objective placement problem has attracted significant research attention, with solutions targeting diverse deployment scenarios and optimization goals. For example, [3] solves the joint user association and service function chain (SFC) placement problem in 5G networks, [5] proposed resource management schemes for Industrial IoT applications in Cloud-Edge networks. Thorough survey of literature surrounding the placement problem in various computing paradigms can be seen in [10]. However, many existing work differ from ours in the underlying assumptions and granularity of problem-solving. For instance, the aforementioned approaches leverage a Kubernetes-based architecture that limits its applicability. Consequently, their application to our context is not feasible.

Existing task offloading research often investigates coarse-level abstractions of applications and overlook the task dependencies, which may be sufficient for jobs with less stringent latency requirements, while a number of approaches employ fine-grained service placement strategies that account for inter-service dependencies and allow more precise control for latency-sensitive applications. [7] extracts the function-level dependencies from the application using analysis tools. [8] studies the offloading of sub-tasks with dependencies and the allocation of communication resources in unmanned aerial vehicles assisted MEC systems. The approaches mentioned above either reduces a single metric, or incorporate specific strategies such as service replication. While in our work, we aim to reduce both the total energy consumption and latency throughout application execution without resorting to service replicas.

Service placement solutions typically target various objectives, including minimising end-to-end latency [2, 3], reducing costs [16], lowering energy consumption [4], and improving resource utilization [13]. In the context of multi-objective optimization problems, researchers have employed a diverse set of approaches, including heuristics, meta-heuristics, and deep reinforcement learning, to achieve a balance between different objectives. For example, [17] proposes a joint placement algorithm for non-scalable services, balancing latency and deployment cost. Our proposed algorithms aim at optimising both user experience and energy efficiency. We strive to minimise end-to-end latency and reduce the dynamic energy consumption caused by computations.Several prior studies have explored optimisation objectives that converge with those targeted by our proposed approach. While work like [15] explore the energy-delay trade-off using monolithic task scheduling, it lacks dependency awareness. More recently, [16] presents a task offloading scheme for dependent tasks, jointly optimising latency and energy. However, their focus is on local execution vs. edge offloading, while ours lever-

ages both Multi-access Edge Computing (MEC) and cloud resources. Similarly, regarding the energy consumption optimisation in dependency-aware placement strategies, [8, 12] primarily focus on minimising energy consumption at the user device level.

Our work departs from the existing literature by considering a more realistic and intricate scenario that incorporates Cloud-MEC network characteristics and inter-service dependencies, and optimising energy consumption across the entire system comprehensively. We propose two heuristic-based algorithms designed for fast execution. These algorithms optimise user-experienced latency and minimise the overall energy consumption associated with task execution, allowing for a holistic optimization of both QoS and environmental sustainability.

3 System model and problem formulation

3.1 MEC network model

Our work considers a multi-tier Cloud-MEC network infrastructure (Figure 1) where micro datacenters (MDCs) act as resource pools at the edge, residing at interconnected mobile stations. Within each MDC, two key entities exist: edge **computing servers** available for service deployment and execution, and **data sources** that provide data flow from databases, sensors, and IoT devices without performing computations locally. Query-requests from various **user devices** can be concurrently handled by the MEC. The multi-access edge network is connected to a resource-rich cloud data center (Cloud DC).

We model the Cloud-MEC network topology as a graph: $G = (M, L)$. $M = \{m_1, m_2, ...m_n\}$ refers to the set of n MDCs and Cloud DC, and $L = \{l_1, l_2, ...l_p\}$ represents the set of network connections between the DCs. The set of computing servers maintained by MDC m_i can be expressed as $S_i = \{s_{i,1}, s_{i,2}, ...s_{i,m}\}$, while $D_i = \{d_{i,1}, d_{i,2}, ...d_{i,k}\}$ indicates the data sources connected to the local network of m_i. Server specifications of each $s_i \in S$ are given by $\{\gamma_{s_i}, \mu_{s_i}, f_{s_i}\}$, representing respectively the memory capacity, storage capacity and CPU frequency. For every network link $l \in L$, the two MDCs connected through it are expressed as (l^{src}, l^{dst}). The link bandwidth of l is denoted by b_l, and the propagation delay of l is a constant value $prop_l$.

3.2 Application model

We use the directed acyclic graph (DAG) to establish the application model. The application A that the user equipment (UE) requests to execute can be represented in the form of a DAG $A = (V, E)$. $V = \{D, O\}$ is the set of nodes composing the application, and E is the directed edge set representing dataflow dependencies between the modules. We let $D = \{d_1, d_2, ...d_m\}$ to represent the data sources involved in the application, and use $O = \{o_1, o_2, ...o_n\}$ to indicate the set of service modules that are responsible for dataflow processing operations. The end user is denoted by U.

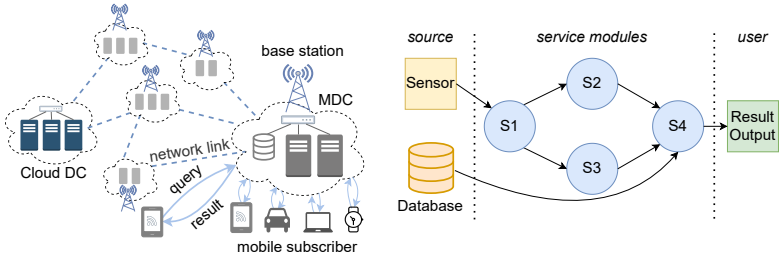

Fig. 1: An example Cloud-MEC network and an example app.

Each service module $o_i \in O$ represents a processing operator or function, with a set of properties $\{\alpha_{o_i}, \theta_{o_i}, \omega_{o_i}\}$ denoting the memory, storage and CPU cycles it requires to process each task. Modules with no predecessors that require input from outer data sources are called entry modules. Similarly, we refer to the modules without successor nodes as exit modules. The outputs of all exit nodes will be directed to the end user. Each edge e in the set $E = \{e_1, e_2, ...e_o\}$ has an attribute δ_e representing the size of each packet transmitted. The start node e^{start} of the directed edge e is the source module sending data packets through the edge, and the end node e^{end} is the destination. Our processing model adheres to the following assumptions: (i) a service is triggered only when it has received inputs from all its predecessors, (ii) each task represents the minimal unit of work and is indivisible, and (iii) each service node sends results to its successors only after finishing its task processing.

3.3 Problem formulation

Communication and computation model. The communication time $T^{comm}_{e,l}$ for a data packet corresponding to DAG edge e to be transmitted through network link l can be calculated by summing up the data transmission time (packet size δ_e divided by bandwidth b_l) and the propagation delay, as expressed in Eq. 1. The deployment plan of a service module o is denoted by $P(o)$, which is a mapping from service module o to server s. For a DAG edge e and its two vertices o_{start}, o_{end}, we assume these two modules are offloaded to servers s_i and s_j respectively. We use $path(s_i, s_j) = \{l_1, l_2...l_n\}$ to denote the routing path from server s_i to s_j. When the predecessor module o_{start} sends a data packet to the successor o_{end}, the communication delay is calculated by summing up the transmission delay and propagation delay along the routing path. Therefore, the total communication delay is obtained by Eq. 2. Additionally, between two services deployed in the same MDC, we assume a constant network delay.

$$T^{comm}_{e,l} = \delta_e / b_l + prop_l \qquad (1) \qquad T^{comm}_{e,s_i s_j} = \sum_{l}^{path(s_i,s_j)} (\frac{\delta_e}{b_l} + prop_l) \qquad (2)$$

Assume that any computing server s_i runs at a constant frequency f_{s_i} when processing tasks, given the workload required by a service module o to process

a data packet requiring ω_o CPU cycles, the execution time can be calculated by Eq. 3.

$$T_{o,s_i}^{exec} = \frac{\omega_o}{f_{s_i}} \tag{3}$$

$$T_{o|n}^{exec} = T_o^{exec} * K(n) \tag{4}$$

For multiple service modules operating concurrently on the same server, the overhead caused by resource contention among the services is introduced by an overhead function K. $K(n)$ denotes the overhead coefficient for the multi-tenancy scenario of n service modules deployed on one server.When n service modules including o are deployed on the same server and sharing resources, the task processing time of o is calculated by Eq. 4.

According to Section 3.2, for service modules with multiple predecessor nodes, the service module will not start the task execution until data packets from all the predecessors have arrived. The waiting time caused during this process affects the user-experienced delay. Therefore, for a service module o, the earliest time it starts processing tasks is decided by its most time-consuming predecessor. For o with its predecessors $pred(o)$, we obtain its *Earliest Start Time (EST)* by:

$$EST(o) = \max_{o_i \in pred(o)} \left\{ EST(o_i) + T_{o_i}^{exec} + T_{o_i,o}^{comm} + T_{o_i,o}^{prop} \right\} \tag{5}$$

Thus the end-to-end latency experienced by user U is $EST(U)$.

Energy consumption model. We drive the dominant, dynamic energy consumption of each server by calculating its dynamic power consumption over time. According to [15] and Eq. 3, assuming the supply voltage of the CMOS circuits and CPU operating frequency are linearly dependent, for a server running a service module o, the dynamic energy consumption during the processing period of a task is approximated by $EC_{dynamic} = \beta f^2 \omega_o$, where β represents a device related factor.

Placement problem formulation. The objective of the present problem is to minimise the user-experienced latency and reduce overall dynamic energy consumption. The solution should be an appropriate mapping from the service nodes to the servers. We call a placement plan valid only if all the service modules are assigned to servers with sufficient resources and all the constraints are met. In light of this, we formulate the conditions that make a placement plan valid: (i)Given the set of service modules O to place and the set of available servers in the Cloud-MEC environment G, each service in O should be mapped to a server in G. We formulate such a progress as a mapping function P, let S denote the set of all servers in the network, then $P : O \rightarrow S$ indicates a complete deployment of all services. (ii)The summation of resource occupancy for all assigned modules should not violate the capacity of the server.

$$\left. \begin{array}{l} \Phi_s = \{o \mid o \in O, P(o) = s\} \\ \forall s \in S \end{array} \right| \Rightarrow \begin{cases} \sum_{o \in \Phi_s} \theta_o \leq \mu_s \\ \sum_{o \in \Phi_s} \alpha_o \leq \gamma_s \end{cases} \tag{6}$$

(iii)Aside from these basic conditions, various optimisation goals may exist. In this work, our focus is to minimise user-experienced latency and overall energy consumption, therefore the multi-objective optimisation problem can be formulated as: **Opt** : $min\ EST(U), min\ \sum_{s \in S} EC_s$.

4 Energy-and-latency-aware placement algorithms

4.1 Energy-aware delay-experienced minimisation algorithm

The proposed service placement algorithm, Energy-aware delay-experienced Minimisation (EDEM), adopts a two-stage approach to achieve a balance between latency minimisation and energy consumption reduction. In the first, coarse-grain stage, EDEM determines a service-to-MDC deployment plan that prioritises reducing end-to-end latency. Subsequently, the fine-grain stage refines the server deployment plan within each MDC, focusing on optimising energy consumption without affecting the overall latency.

Coarse-grain scheduler: The critical-path-based coarse-grain scheduler seeks a balanced configuration with minimal transmission latency and maximized processing efficiency, leading to the lowest overall end-to-end latency. The pseudo-code of the coarse-grain scheduler is presented in Algorithm 1. We use $S(A)$ to denote the state space of services in application A, consisting all service-to-MDC placement options. The scheduler explores $S(A)$ by post-order traversing all placement options of the service modules in A. According to Eq.5, the calculation of EST for any service v rely on the EST value of all its predecessor modules $v.predecessors$. Following such rule, the scheduler can estimate the EST for each service, paving the way for critical-path selection.

The critical path selection works in a greedy and optimistic fashion. Starting from the bottom, for each visited service v, given the set of available MDCs M, the value of $EST(v)$ will be estimated assuming v resides at the least-loaded server $m.leastloaded$ in each MDC $m \in M$. For each predecessor $v.pred$ of v, knowing its estimated EST values $[EST(v.pred, m_1), EST(v.pred, m_2), ...]$, the pair $(v.pred, m)$ that produces the minimum estimated EST will be selected. Among all the selected predecessor-MDC pairs, the one with the maximum EST value will be chosen and marked as a critical node, and the criticality of non-critical modules becomes the EST value difference between it and the critical node in the same hierarchy. Such operation will be repeated until the exit node is reached. Thus we get the critical path reflecting the MDC allocation plan for the critical modules, while non-critical modules remain unassigned. At this stage, the load status of the MDCs involved in the MDC allocation plan will be updated, so is the least-loaded server $m.leastloaded$ of each MDC $m \in M$. Then the max-min procedures will be resumed for the non-critical modules, until their MDC allocation plan is decided.

Fine-grain scheduler: Having the service-to-MDC placement plan decided in the coarse-grain stage, for each MDC involved, a fine-grain scheduler runs to drive the module-to-server placement solution. The fine-grain scheduler tries to place modules on less power-consuming servers, under the premise that the end-to-end latency of critical nodes will not be affected. Critical modules will take priority in placement, followed by non-critical modules. The pseudo-code of the fine-grain scheduler is presented as Step 3 in Algorithm 1. Similar to the design of the coarse-grain scheduler, all service-to-server placement options will be considered. The scheduler estimates the earliest start time of every service

module regarding each available server in the local MDC. Holding the rule that the EST value of modules on non-critical branches should not exceed the EST of critical modules on the same hierarchy, for each possible placement option (v, s), we calculate the value of $EST(v, s)$, and remove the ones that violate the rule. The remaining placement options are marked as 'valid', and the one with the minimum energy consumption $EC(v, s)$ will be chosen and added to the placement plan. The energy consumption will be approximated according to 3.3. Once a service placement decision is finalised, the load status of the server involved will be updated, and the estimated EST of affected placement options will be recalculated. Such process will be repeated until all the service modules are mapped to exact servers.

Algorithm 1: EDEM

Data: App. $A = (V, E)$, MEC $G = (M, L)$
Result: Server placement map $P : O \rightarrow S$
1. Coarse-grain stage:
Initiate $S(A)$ $\forall v \in V, \forall m \in M$; $CP = \emptyset$;
Explore $S(A)$ using post-order traversal:
for $v.pred \in v.prdecessors$ **do**
  for $m \in M$ **do**
    Compute *EST(v.pred,m)*;
  end
  Select *(v.pred,m)* with *min EST(v.pred,m)*;
end
CP $\leftarrow$ *(v.pred,m)* with *max(EST(v.pred,m))*;
Compute *Criticality(v)*;
2. Fine-grain stage:
for $(v, m) \in critical\_path$ **do**
  Initiate $S(V)$ $\forall s \in m.servers$;
  for $s \in m.servers$ **do**
    if $EST(v, s) \leq Criticality(v)$ **then**
      Compute *EC(v,s)*;
  end
  Assign v to s with *min EC(v,s)*;
  Update load status of s;
end
for $(v, m) \notin critical\_path$ **do**
  Initiate $S(V)$ $\forall s \in m.servers$;
  for $s \in m.servers$ **do**
    if $EST(v, s) + T_{comm}(v, v.succ) \leq Criticality(v.succ)$ **then**
      Compute *EC(v,s)*;
  end
  Assign v to s with min EC(v,s);
  Update load status of s;
end

4.2 Delay-aware energy minimisation algorithm (DEM)

DEM prioritizes energy efficiency by first seeking an energy-saving server placement plan. Unlike EDEM, which prioritizes latency first, DEM focuses on minimizing energy consumption in the initial stage. This is followed by a refinement stage that fine-tunes the placement plan to optimize latency without exceeding the established energy constraints. DEM can be broken down into the 3 steps:
Step 1: All available servers in the MEC network are sorted by the device-related energy consumption coefficient, in ascending order. Then a level-order traversal of the application graph starts from the entry modules. At each layer, modules are randomly assigned resources from the pre-ranked server list. After the assignment, DEM estimates the EST value for each module, using Eq. 5.

Step 2: While ***Step 1*** prioritizes energy-efficient servers, it might not guarantee minimal total energy consumption. This is because geographically distant placements of dependent modules can lead to longer data transmission times and increased server idle energy usage while waiting for packets. To address this, a refinement stage iteratively explores alternative placements for each service module. For each module, all unexplored MDCs (excluding its current location) are considered. Within each unexplored MDC, the least power-consuming server is evaluated for potential reassignment. The algorithm estimates the total energy

consumption EC after each potential reassignment and updates the placement plan if a more energy-efficient configuration is found. Finally, the refined placement's overall latency $EST(U)$ and total energy consumption $\sum_{s \in S} EC(s)$ are calculated and stored for the next stage.

Step 3: Building upon the refined resource allocation (***Step 2***), DEM also focuses on improving user-experienced latency. Similar to EDEM's fine-tuning, it identifies critical modules through the application's critical path. For each critical module, it explores reassignment to alternative servers intending to reduce the critical path's estimated latency. After each reassignment, the critical path is recalculated to reflect potential latency improvements. This iterative process continues until no further latency reduction is possible. The resulting placement plan, balancing energy efficiency and latency, is then used to generate the final module-to-server mapping. The pseudo-code of DEM is presented in Algorithm 2.

Algorithm 2: DEM

Data: Application $A = (V, E)$, MEC network $G = (M, L)$
Result: Server placement map $P : O \rightarrow S$
1. Server sorting and initial service assignment:
Sort all servers $s \in G$ by $coeff(s)$;
$curServerIdx \leftarrow 0$;
Group service from level n to 0 and shuffle each set $S_n, ...S_0$
for *service* $v \in S_n, ..., S_0$ **do**
    Assign v to server with index $curServerIdx$;
    $curServerIdx{+}{+}$;
end
for $v \in V$ **do**
    Compute $EST(v,P(v))$ using Eqs;
end
2. Energy-aware reassignment:
for $v \in V$ **do**
    for $m \in M$ **do**
        if $P(v) \notin m$ **then**
            s = $\arg\min_{s' \in m.\text{servers}} \text{coeff}(s')$;
            if $EST(v,s) < EST(v,P(v))$ **then**
                Assign server s to v;
    end
end
Compute $EST(U)$ using Eqs;
Sum up $EstimatedEC(v, P(v)) \; \forall \; v \in V$;
3. Latency-aware reassignment:
Identify $critical\_path$ of A under placement P;
for $(v, P(v)) \in critical\_path$ **do**
    for $s \in P(v).mdc.servers$ **do**
        if $EST(U) > EST(U|P(v) = s)$ **then**
            Assign server s to v;
            Re-identify $critical\_path$ of A;
    end
end

5 Experimental Evaluation

5.1 Performance Indicators & Setup

We employed simulation to evaluate the performance of the proposed algorithms (DEM and EDEM). The YAFS fog simulator [6] was used and extended to support sequential processing of dependent tasks. Network topologies were created using the NetworkX library[5]. These topologies consisted of up to 20 datacenters, one acting as the cloud and the others as micro edge datacenters. Each datacenter housed up to 5 computing servers. Details regarding the specific MEC network configurations are provided in Table 1. Three different random graph generation models were utilized: Barabasi-Albert (B-A) [1], Watts-Strogatz (W-S) [14], and ring topology. To achieve more general results, real-world workloads were utilized from the Alibaba[6] cluster trace dataset. This dataset provides Directed Acyclic Graph information of production batch workloads from a large-scale cluster.

[5] `https://networkx.org/`
[6] `https://github.com/alibaba/clusterdata`

Jobs composed of computing and algorithm services as well as statistical and data processing services, were submitted by users. We filtered applications exceeding 10 modules and selected 10 for evaluation (Table 2). Module resource requirements were configured based on the provided traces. For each experiment set, system events were simulated for 20,000 global timestamps and repeated 5 times with identical configurations to generate statistically significant averages. The simulations were conducted on a server with 4x Intel Xeon Gold 6230N CPU, 256GB of RAM and Ubuntu 20.04 operating system.

The proposed algorithms (DEM and EDEM) were evaluated based on four key metrics: i) Overall Energy Consumption (EC): This metric represents the total power consumed by all servers during the execution period, estimated from CPU usage data according to 3.3. ii) Average User-Experienced Latency (LT): This metric is the mean response time of user requests, calculated from raw timestamps recorded by the simulator during network transmissions. iii) Edge prioritization (EP): This metric reflects the percentage of services deployed at the Edge, iv) Algorithm Execution Time (ET): This metric indicates the time required for the algorithm to generate a placement, i.e. the wall clock time for executing each algorithm. The performance of DEM and EDEM was compared against four existing algorithms: Response Time Aware (RTA) [2], Genetic Algorithm (GA) [11], Maximize Reliability Offloading (MROA) [12] and Energy-Makespan Multi-objective Optimization(EM-MOO) [4]. We selected these algorithms for their varying complexity, computational overhead, and optimization goals, providing a broader assessment of DEM and EDEM's efficiency.

	Cloud	**MDC**
CPU frequency (GHz)	[3,5]	[1,2]
RAM (GB)	[32,64]	[1,4]
Propagation Delay (ms)	8	1
Bandwidth (Gbps)	10	2

Table 1: Configurations

id	$\|V\|$	$\|E\|$	Max degree	Average transfer volume	Average workload
0	12	9	3	39.33	9.50
1	16	17	6	29.00	257.88
2	16	17	7	23.63	40.50
3	17	17	5	42.82	13.53
4	16	17	3	46.06	35.75
5	12	11	6	38.67	8.17
6	10	10	4	44.30	14.40
7	10	9	5	35.60	7.10
8	16	16	2	47.19	1.00
9	10	9	4	43.40	32.70

Table 2: Application characteristics

5.2 Performance Assessment

In total, we conducted 8100 experiments, consisting of 5 experiments for each of the 6 algorithms and for each of the 10 applications. We varied the number of MDCs (denoted by n) between [5, 10, 20], and the number of servers (denoted by m) within each MDC to [2, 4, 8]. Also, we utilized three different network topologies. All figures presented in this section demonstrate the normalized performance of both EC and LT metrics. We set the source data emission interval to 100ms and the simulated time to 100s and plot the performance of the proposed algorithms in terms of the aforementioned metrics. In Fig. 2, we show the normalized energy consumption for all algorithms. It can be seen that during the experiments using varied applications and under varied network typology, the DEM algorithm achieves significantly lower energy consumption

compared to other algorithms, while RTA and MROA reaches the highest energy consumption. This is because RTA merely searches for the placement plan that may shorten the end-to-end latency, therefore less energy-efficient servers are chosen, and the overall energy consumption is sacrificed; MROA focuses merely on lowering the energy consumption of the user equipment, rather than that of the servers involved in task offloading, therefore the total energy consumed for task processing is rather high. DEM, on the contrary, tends to allocate services to edge servers with higher energy efficiency. Follwing DEM, EDEM also achieves energy consumption levels comparable to those of the GA and EM-MOO. Across diverse network topologies, DEM and EDEM maintain stable performances in terms of normalised energy consumption, while EM-MOO falter under the Watts–Strogatz network model. This proves that DEM and EDEM are robust to network variations.

Fig. 3 demonstrates the normalized latency. We can observe that RTA, EDEM and MROA reach the lowest latency under all the scenarios. It should be highlighted that RTA and MROA sacrifice energy efficiency to achieve their performance, whereas EDEM maintains metrics. GA tends to demonstrate the worst performance for all network topologies while EDEM excels in certain applications. Specifically, in the ring topology, EDEM provides better latency performance which is crucial when network latency poses a significant bottleneck. Combining the results from Fig. 2 and Fig. 3, we can state that, compared to existing works, DEM and EDEM hold different degrees of preference for reducing energy consumption and lowering latency. EDEM effectively balances overall energy consumption and latency. On the other hand, DEM prioritizes energy-saving placement plans, potentially sacrificing latency to some extent.

Figs 4, 6 reveal that for the EC metric, the DEM and GA algorithms show the most significant improvements and efficiency when increasing either the number of MDCs or servers, while MROA consistently underperforms. EMMOO shows a slight decrease in energy consumption as the number of resources increases. Although EDEM is not the best choice, it demonstrates stable performance, making it a reasonable option. For the LT metric, RTA, EMMOO, and DEM perform better when increasing the number of MDCs, with EDEM following closely behind (Fig. 5). GA and MROA struggle with scalability in reducing latency as the system becomes more complex. In Fig 7, we observe that increasing the number of servers generally worsens latency performance for most algorithms. Although GA shows some improvement as the number of servers increases, it still cannot effectively optimize latency as the system scales up, compared to the other algorithms. EDEM, similar to its performance with the EC metric, remains stable, making it the best candidate when the number of servers exceeds four. Table 3 offers a breakdown of service deployment across edge servers and the cloud. The number shown in the table represents the percentage of services deployed at the edge. The results of MROA align with our analysis: it greedily selects non-local, powerful machines. Consequently, around 40% of the services are offloaded to the cloud, resulting in the highest energy consumption observed. RTA and EDEM have similar preferences in adopting resources. Around 75-85% of the services

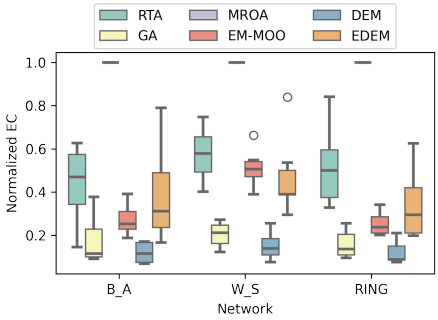

Fig. 2: EC, n=20 m=4

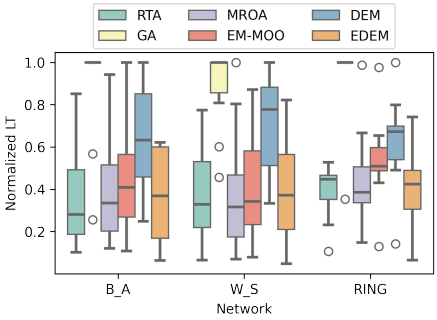

Fig. 3: LT, n=20 m=4

are deployed at edge facilities, with the remaining tasks executed on the cloud. Such approach achieves a well-balanced outcome in terms of both latency and energy consumption: performing most computations at the edge helps to reduce transmission and energy costs, while offloading a select few, computationally intensive tasks to the cloud minimises processing time. By deploying over 95% of services at the edge, DEM, EMMOO and GA demonstrate the strongest tendency to utilise edge resources. Prioritising edge placement leads to the lowest energy consumption, but as a consequence of the energy-latency trade-off, the task execution time may not be as low as cloud execution, potentially sacrificing user-experienced latency.

The execution time of each algorithm is presented in Table 4. As expected, GA exhibit the longest execution times (∼4s), followed by EMMOO (∼1s), due to their population-based search and iterative nature, respectively. Conversely, MROA achieves the fastest execution times across all topologies because its operations related to latency and energy consumption limitations are skipped when no deadline constraints are set. RTA achieves the second fastest execution times across all topologies as it focuses solely on a single objective. EDEM and DEM demonstrate execution times comparable to RTA, thus can be adopted to respond in real-time and time-sensitive applications at the MEC. We also conducted an additional set of experiments to assess the impact of a MEC environment. In these experiments, we assumed a setup with 4 MDCs, 1 Cloud DC, and a BA topology. The results show that, on average, latency is reduced by 46.8% and energy consumption by 32.9% when scheduling decisions select resources from the edge-cloud continuum instead of relying solely on cloud resources.

6 Conclusions and future work

This paper introduces EDEM and DEM, two algorithms for service module placement in MEC networks that consider dependencies between service modules. EDEM prioritizes energy efficiency while maintaining low latency impact

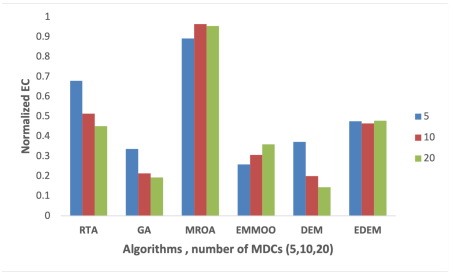

Fig. 4: EC, n=[5,10,20]

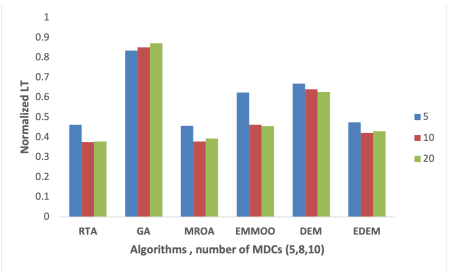

Fig. 5: LT, n=[5,10,20]

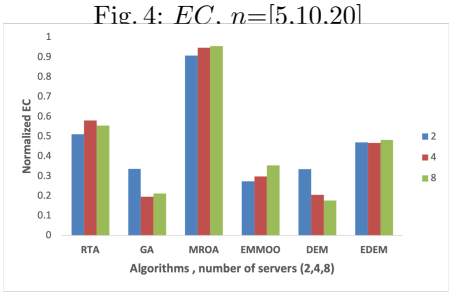

Fig. 6: EC, m=[2,4,8]

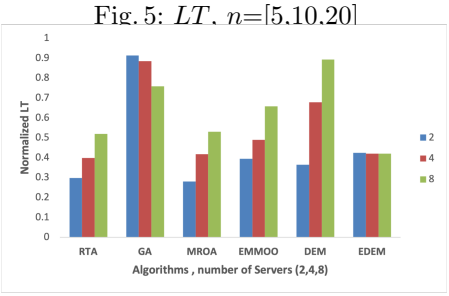

Fig. 7: LT, m=[2,4,8]

	RTA	GA	MROA	EMMOO	DEM	EDEM
B-A	85.86	97.75	58.52	98.80	98.24	85.70
RING	76.61	89.79	57.90	99.66	95.19	84.24
W-S	76.85	89.69	60.40	99.13	94.04	84.46

Table 3: Edge prioritization, EP

	RTA	GA	MROA	EMMOO	DEM	EDEM
B-A	45.35	4110.22	3.13	1077.88	224.91	138.63
RING	42.79	4475.50	3.37	1066.64	315.66	173.59
W-S	45.96	4407.99	3.15	1064.44	290.66	161.51

Table 4: Execution time (ms), ET

on users. DEM, on the other hand, achieves significant energy reductions by allowing for a more flexible trade-off with increased latency. Factors like the queuing of queries from different users that can impact the overall transmission delay, will be addressed in the next step. Future work will also incorporate user mobility into the design for better real-world applicability. Additionally, we aim to develop online versions of EDEM and DEM, enabling dynamic service rescheduling based on changing resource availability in MEC networks.

7 Acknowledgement

This work was supported in part by the Research Institute of Trustworthy Autonomous Systems (RITAS), and in part by the Shenzhen Science and Technology Program (project No. GJHZ20210705141807022).

References

1. Barabási, A.L., Albert, R.: Emergence of scaling in random networks. science **286**(5439), 509–512 (1999)

2. Cai, X., Kuang, H., Hu, H., Song, W., Lü, J.: Response time aware operator placement for complex event processing in edge computing. In: International Conference on Service-Oriented Computing. pp. 264–278. Springer (2018)
3. Harutyunyan, D., Shahriar, N., Boutaba, R., Riggio, R.: Latency and mobility–aware service function chain placement in 5g networks. IEEE Transactions on Mobile Computing **21**(5), 1697–1709 (2020)
4. Ijaz, S., Munir, E.U., Ahmad, S.G., Rafique, M.M., Rana, O.F.: Energy-makespan optimization of workflow scheduling in fog–cloud computing. Computing **103**, 2033–2059 (2021)
5. Kaur, K., Garg, S., Kaddoum, G., Ahmed, S.H., Atiquzzaman, M.: Keids: Kubernetes-based energy and interference driven scheduler for industrial iot in edge-cloud ecosystem. IEEE Internet of Things Journal **7**(5), 4228–4237 (2019)
6. Lera, I., Guerrero, C., Juiz, C.: Yafs: A simulator for iot scenarios in fog computing. IEEE Access **7**, 91745–91758 (2019)
7. Mo, J., Liu, J., Zhao, Z.: Exploiting function-level dependencies for task offloading in edge computing. In: IEEE INFOCOM 2022-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). pp. 1–6. IEEE (2022)
8. Nguyen, L.X., Tun, Y.K., Dang, T.N., Park, Y.M., Han, Z., Hong, C.S.: Dependency tasks offloading and communication resource allocation in collaborative uavs networks: A meta-heuristic approach. IEEE Internet of Things Journal (2023)
9. Oikonomou, P., Karanika, A., Anagnostopoulos, C., Kolomvatsos, K.: On the use of intelligent models towards meeting the challenges of the edge mesh. ACM Computing Surveys (CSUR) **54**(6), 1–42 (2021)
10. Salaht, F.A., Desprez, F., Lebre, A.: An overview of service placement problem in fog and edge computing. ACM Computing Surveys (CSUR) **53**(3), 1–35 (2020)
11. Sarrafzade, N., Entezari-Maleki, R., Sousa, L.: A genetic-based approach for service placement in fog computing. The Journal of Supercomputing **78**(8), 10854–10875 (2022)
12. Shang, Y., Li, J., Wu, X.: Dag-based task scheduling in mobile edge computing. In: 2020 7th International Conference on Information Science and Control Engineering (ICISCE). pp. 426–431. IEEE (2020)
13. Skarlat, O., Nardelli, M., Schulte, S., Borkowski, M., Leitner, P.: Optimized iot service placement in the fog. Service Oriented Computing and Applications **11**(4), 427–443 (2017)
14. Watts, D.J., Strogatz, S.H.: Collective dynamics of 'small-world'networks. nature **393**(6684), 440–442 (1998)
15. Zhang, G., Zhang, W., Cao, Y., Li, D., Wang, L.: Energy-delay tradeoff for dynamic offloading in mobile-edge computing system with energy harvesting devices. IEEE Transactions on Industrial Informatics **14**(10), 4642–4655 (2018)
16. Zhang, Y., Chen, J., Zhou, Y., Yang, L., He, B., Yang, Y.: Dependent task offloading with energy-latency tradeoff in mobile edge computing. IET Communications **16**(17), 1993–2001 (2022)
17. Zhao, X., Shi, Y., Chen, S.: Maesp: Mobility aware edge service placement in mobile edge networks. Computer Networks **182**, 107435 (2020)