Latency-Aware Task Offloading Mechanism for Mobile Edge Computing

Abdulelah Alwabel
Department of Computer Sciences
Prince Sattam Bin Abdulaziz University
AlKharj, Saudi Arabia
e-mail: {a.alwabel}@psau.edu.sa

***Abstract*—Task offloading in Mobile Edge Computing (MEC) is a critical mechanism that enables resource-constrained mobile devices to delegate computational tasks to proximate edge servers for processing. One of the core benefits of task offloading in MEC is the significant reduction in latency. Unlike traditional cloud computing, where data must travel to remote data centers for processing, MEC leverages edge servers positioned at the periphery of the network, close to end users. However, task offloading is not without its challenges. Critical issues, including network reliability, security, data privacy, and effective task scheduling, must be addressed to provide efficient offloading processes. This paper presents a novel latency-aware task offloading mechanism for MEC environments. The proposed mechanism dynamically adapts to latency variations to optimize task placement and migration across edge servers. Unlike traditional approaches, the mechanism operates without prior knowledge of task characteristics, enabling real-time task submission and execution. The mechanism employs a migration policy that minimizes latency by reassigning tasks in the waiting queue based on latency changes caused by mobile device movement, reducing the negative impact of such variations on system performance. To evaluate the effectiveness of the mechanism, a simulation environment was developed to model MEC scenarios. The simulation considered varying task loads and dynamic latency conditions to emulate real-world operations. Results demonstrate that the proposed mechanism achieved an improvement in key performance metrics, including latency, waiting time, and makespan time.**

Keywords-task offloading; dynamic mechanism; latency-aware; MEC.

I. INTRODUCTION

Task offloading in Mobile Edge Computing (MEC) is a critical mechanism that enables resource-constrained mobile devices to delegate computational tasks to proximate edge servers for processing. This approach can bridge the gap between the resource limitations of mobile devices and the increasing computational demands of modern applications, such as Augmented Reality (AR) and video processing [1]. By offloading tasks to edge servers located closer to the end user, MEC reduces latency, enhances energy efficiency, and ensures better utilization of computational resources, thus paving the way for seamless user experiences and efficient system operations.

The proliferation of smart devices and the emergence of data-intensive applications have introduced new challenges in mobile computing. Mobile devices, while portable and versatile, often suffer from limited battery life, computational power, and storage capacity. Task offloading addresses these challenges by transferring computational tasks to edge servers, which are equipped with greater processing power and are located at the network edge, closer to users. This minimizes the delay caused by communication with distant cloud servers and alleviates the burden on mobile devices, thereby extending their operational lifespan and improving their performance [2].

Task offloading in MEC is typically categorized into full offloading and partial offloading [3]. In full offloading, the entire computational task is sent to the edge server, leaving the mobile device to act primarily as an input/output interface. This is especially beneficial for highly complex applications where local execution is infeasible due to resource constraints. Partial offloading, on the other hand, involves splitting the task into smaller components, with some parts processed locally and others offloaded. This approach is ideal for tasks that can be parallelized or for scenarios where network conditions or server availability may not support full offloading.

The process of task offloading in MEC involves several key components, including task partitioning, offloading decision-making, and resource allocation [4]. Task partitioning determines how the task is divided into smaller subtasks, while offloading decisions are made based on parameters such as network bandwidth, device resources, latency requirements, and energy consumption. Resource allocation ensures that edge servers have the capacity to handle offloaded tasks efficiently without overloading the system.

One of the core benefits of task offloading in MEC is the significant reduction in latency. Unlike traditional cloud computing, where data must travel to remote data centers for processing, MEC leverages edge servers positioned at the periphery of the network, close to end users. This proximity reduces the round-trip time for data transmission, enabling real-time processing and low-latency responses [5]. Furthermore, by shifting computational tasks away from mobile devices, task offloading helps conserve battery life, a critical consideration for mobile users.

However, task offloading in MEC is not without its challenges. Critical issues, including network reliability [6], security [7], data privacy [8], and effective task scheduling, must be systematically dealt with to enable seamless and secure offloading processes. Furthermore, dynamic environmental factors, such as fluctuating network conditions and varying server workloads, necessitate the development of adaptive and intelligent offloading strategies. In this study, we propose a novel task offloading mechanism that incorporates awareness of dynamic network conditions to optimize the placement and migration of tasks across edge servers. The proposed approach aims to minimize latency, waiting time, and makespan, thereby

enhancing overall system efficiency.

The reminder of this paper is organized as follows. Section II presents related works. Our mechanism is proposed and discussed in Section III. The results of employing our novel mechanism is presented and analyzed in Section IV. The paper concludes with future directions for research in Section V.

II. Related work

Task offloading in edge computing has been extensively explored, particularly in the context of IoT and its impact on network efficiency. For instance, the study in [9] discusses the significant traffic generated by real-time data management in edge networks with full offloading capacity. The authors propose an algorithm to detect node faults, manage deadlines, and improve data handling efficiency in centralized systems, reducing bandwidth use and scheduling delays.

Edge and cloud server performance comparisons by the authors in [10] highlight the efficiency of edge servers in resource utilization, while cloud servers excel in cost and delay reduction. To handle offloading inefficiencies, they propose an algorithm, which minimizes delays, optimizes resource allocation, and enables parallel task execution, enhancing system responsiveness.

The use of heuristic algorithms is detailed by the researchers in [11], who introduce an approach based greedy policy for resolving task offloading challenges. It integrates MEC to address latency issues in computation-intensive tasks, optimizing task management and resource efficiency while mitigating battery life concerns. However, this work pays little attention to changes of latency during run time.

Decentralized architectures for edge computing are explored in [12], which introduces a hierarchical edge cloud model. This architecture addresses limitations of traditional cloud computing, improving scalability, fault tolerance, and data recovery, particularly for applications requiring high mobility and low latency.

Improved Quality of Service (QoS) in edge computing is a recurring theme. For instance, the authors in [13] advocate for predictive systems using collaborative filtering to prevent delays. The results of this study demonstrate that QoS can be improved using machine learning approaches. In addition, the study in [14] proposes a reliable pooling approach to address task distribution and overhead costs. These approaches enhance system reliability and optimize resource usage.

A novel model for resource-efficient edge computing tailored for smart IoT applications is introduced [15]. A hybrid device-based computation offloading method was developed to optimize resource usage. The primary objective of this research is to enable diverse smart IoT device users to minimize cloud resource consumption while adhering to QoS constraints. A key advantage of the proposed approach lies in its ability to enhance the algorithm's performance, particularly in terms of resource efficiency [16].

The authors in [17] propose a dynamic time-sensitive scheduling algorithm that integrates the First-Come, First-Served (FCFS) policy with priority-aware scheduling. However, the proposed mechanism assumes prior knowledge of tasks to enable prioritization based on their deadline requirements.

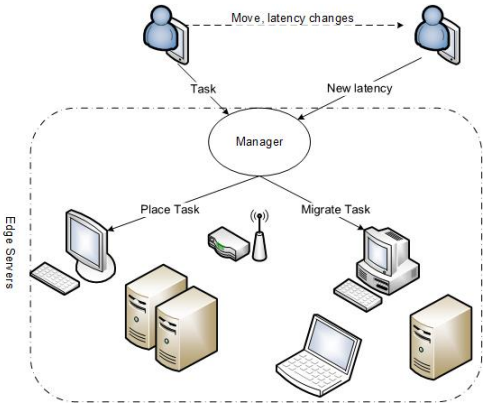

Figure 1. System Model

Overall, the literature highlights the potential in MEC to mitigate latency, optimize task scheduling, and enhance fault tolerance. However, challenges such as dynamic network conditions necessitate further research to refine adaptive and intelligent offloading mechanisms.

Algorithm 1 Initial Placement - DM Mechanism

1: get $task$, $nodeList$
2: $mkn_{min} = maximumValue$
3: **foreach** $node$ in $nodeList$ **do**
4: mkn_{tmp} = Makespan($task$, $node$) //equation 1
5: **if** $mkn_{tmp} < mkn_{min}$ **then**
6: $mkn_{min} = mkn_{tmp}$
7: $node_{selected} = node$
8: **end if**
9: **end for**
10: place($node_{selected}$,$task$)
11: update $nodeList$

III. Proposed Mechanism

This section introduces a Dynamic Mechanism (DM) designed to offload tasks from devices to edge servers within the MEC environment. The proposed mechanism is aware latency variations during runtime that dynamically adapting assigns and migrates tasks without prior knowledge of tasks before their submission with an aim to improve performance. The mechanism allows tasks to be submitted at any point during runtime.

Figure 1 illustrates the system model of a Mobile Device (MD) that offloads a task to a manager module within the MEC environment. The manager finds an edge server, to be called node, to execute this task. This selection plays an important rule to improve the QoS of the system. Algorithm 1 shows an initial placement process to select a suitable node ($node$) to host a recently submitted task ($task$). The process selects a

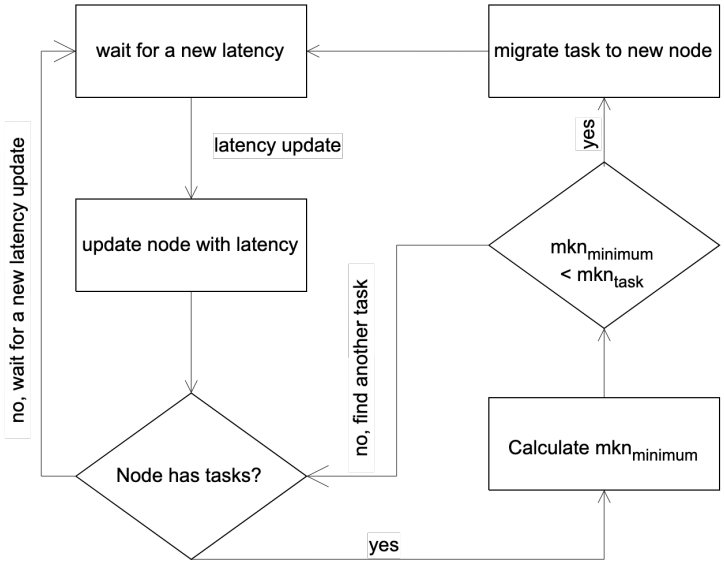

Figure 2. Migrate Process - DM Mechanism

node with a minimum makespan time (mkn). It is calculated as:

$$mkn(task, node) = l + wt(node) + et(task, node) \quad (1)$$

where l denotes the latency of this node which means the time required to send the response from this node to the MD which offloaded. $wt(node)$ refers to the total waiting time for this $task$ before it can be executed, it is calculated as:

$$wt(node) = \frac{\sum_{i=1}^{t}(task_t.length)}{node_{cpu}} \quad (2)$$

where $node_{cpu}$ refers to the processing power of $node$. It is measured in Million Instructions Per Second (MIPS) [18]. $et(task, node)$ denotes the execution time and is given as [19]:

$$et(node) = \frac{task.length}{node_{cpu}} \quad (3)$$

$task.length$ refers to the processing length of a task which is measured in Million Instructions (MI) [20]. In our system, we assume that when an MD moves from one location to another, the latency between the MD and nodes changes (i.e., latency can increase or decrease). This change in latency can significantly impact the overall makespan time of tasks. To address this, the DM mechanism incorporates a migration policy designed to account for latency variations. The migration policy is illustrated in Figure 2. When an MD relocates, the manager module is updated with the MD's new location, and it subsequently updates all nodes with the recent latency changes.

The DM mechanism then calculates the mkn value for each task in the waiting list across all nodes. If a node is identified that can execute a task with a lower mkn than the current assigned node, the manager migrates the task to that node. It is important to note that the DM mechanism applies this migration policy only to tasks that have not yet started execution (i.e., tasks in the waiting list). Tasks already in execution are excluded from this process in order to avoid the impact of migration overhead.

TABLE I. SIMULATION CONFIGURATION

Parameter	Value
Latency:	
Initial value	1-10 ms
Value during runtime	1-100 ms
Update time	1 - 100 ms
Node Specifications:	
RAM	128 MB
CPU	1GB, 1.5GB, 2 GB and 2.5GB
Number of nodes	10
Tasks:	
Processing length	(5, 10, 20, 100) $\times$ 10^3 MIPS
Number of Tasks	1000

IV. EVALUATION

This section presents the results obtained from testing the proposed mechanism in a simulation environment. It begins with a discussion of the simulation settings and concludes with an analysis of the results.

A. Experiment Configurations

We extended the simulation tool DesktopCloudSim [21] to simulate the MEC environment. DesktopCloudSim, which is based on the widely-used cloud simulation framework

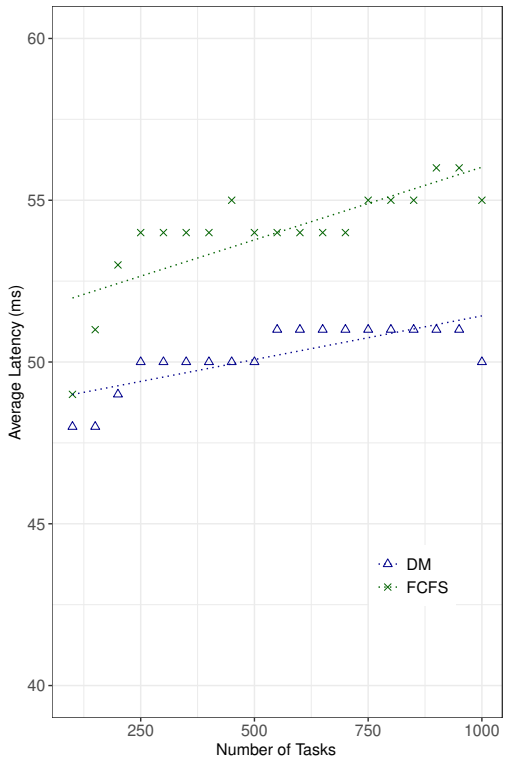

Figure 3. Average Latency

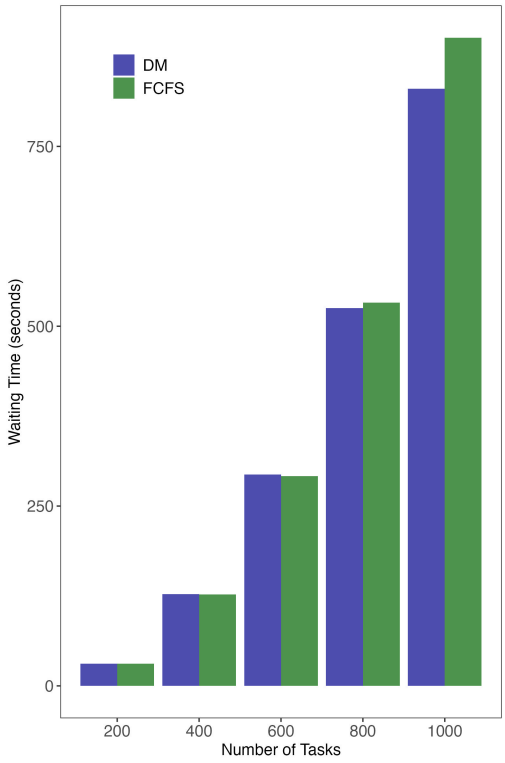

Figure 5. Waiting Time

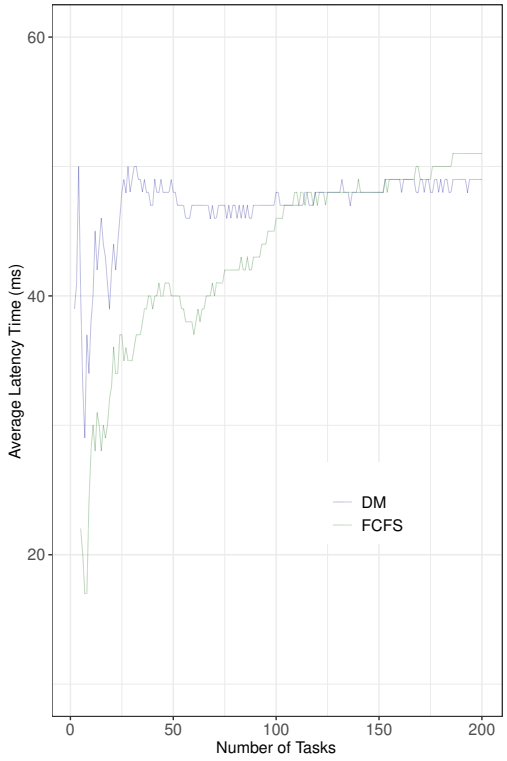

Figure 4. Average Latency (Tasks < 200)

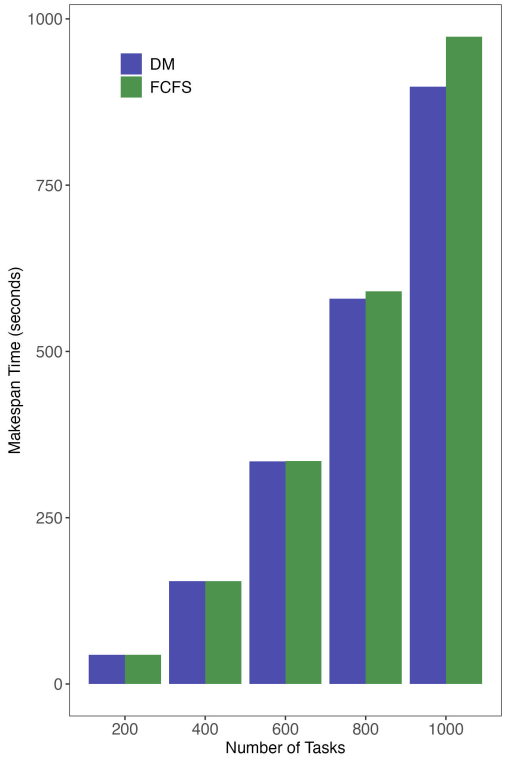

Figure 6. Makespan Time

TABLE II. RESULTS SUMMARY.

Metric	DM Mechanism	FCFS Mechanism
Latency	50 ms	53 ms
Waiting Time	830 ms	938 ms
Makespan Time	898 ms	1010 ms

CloudSim [22], was selected due to its adaptability and extensibility for edge computing scenarios. To evaluate the performance of the proposed mechanism, the simulation was configured with detailed specifications for tasks and edge servers, ensuring an accurate representation of the MEC environment.

Table I presents the configuration of the simulation. It details the initial latency between MDs and nodes, where the latency varies randomly during runtime to simulate the movement of MDs between different locations. The table also specifies the configuration and number of nodes utilized in this study, as well as the total number of tasks submitted to evaluate the DM. The experiment begins with a single task and incrementally increases the number of tasks in each run with one, then two, and continuing up to 1,000 tasks. The length of each task is assigned randomly and is measured in MIPS.

To minimize measurement errors in our simulation tools, we generated tasks randomly, as stated in Table I. These tasks were integrated into the simulation tool in exactly the same way for all evaluated mechanisms. Furthermore, we used a large dataset of 1,000 tasks to further reduce measurement errors in the simulation.

B. Results

Figure 3 demonstrates that the DM mechanism outperformed the FCFS mechanism in terms of average latency as the number of tasks exceeded 200. However, for task counts below 200, the FCFS mechanism performed better than the proposed DM mechanism, as illustrated in Figure 4. This behavior can be attributed to the smaller number of tasks, resulting in fewer tasks in the waiting queue before the MD moves. Since the DM mechanism migrates tasks in the waiting list based on latency changes, any variation in node latency for a task in execution leads to significantly higher latency.

Regarding waiting time, the average waiting time for tasks was approximately 901 ms under the FCFS mechanism, compared to about 829 ms under the DM mechanism. Figure 5 highlights the trend of increasing waiting time as the number of tasks grows.

For makespan time, the DM mechanism achieved an average of 472 ms per task, whereas the FCFS mechanism recorded an average of 487 ms. These results indicate that the DM mechanism reduced the makespan time by approximately 3%. Figure 6 presents a comparison of the makespan results for both mechanisms across the experiment.

Table II presents a summary of the results of this paper, comparing the DM and FCFS mechanisms in terms of latency, waiting time, and makespan time (average values). The results indicate that the DM mechanism consistently outperforms FCFS across all three metrics, demonstrating superior efficiency in task scheduling.

V. CONCLUSION AND FUTURE WORK

Task offloading in MEC presents significant challenges, particularly in managing network reliability and latency variations. To address these issues, this paper proposed a novel task offloading mechanism that dynamically incorporates latency awareness to optimize task placement and migration across edge servers. The experimental results demonstrated that the proposed mechanism effectively reduces latency, waiting time, and makespan compared to the FCFS mechanism, thereby improving overall system performance.

The future directions for this research are twofold. First, the mechanism will be extended to support the migration of tasks that have already commenced execution. This extension will require a comprehensive evaluation of the associated overheads and their impact on system performance. Second, the focus will shift toward exploring additional performance factors, such as load balancing and power consumption of edge servers. These aspects are crucial for enhancing the scalability and energy efficiency of MEC systems.

REFERENCES

[1] B. Zhou, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, "An Online Algorithm for Task Offloading in Heterogeneous Mobile Clouds," *ACM Transactions on Internet Technology*, vol. 18, no. 2, pp. 1–25, Jan. 2018. DOI: 10.1145/3122981.

[2] P. Mach and Z. Becvar, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017, ISSN: 1553-877X. DOI: 10.1109/COMST.2017.2682318.

[3] F. Saeik *et al.*, "Task offloading in Edge and Cloud Computing: A survey on mathematical, artificial intelligence and control theory solutions," *Computer Networks*, vol. 195, p. 108 177, Aug. 2021. DOI: 10.1016/j.comnet.2021.108177.

[4] W. Tang, S. Li, W. Rafique, W. Dou, and S. Yu, "An Offloading Approach in Fog Computing Environment," in *2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, IEEE, Oct. 2018, pp. 857–864. DOI: 10.1109/SmartWorld.2018.00157.

[5] A. Pakmehr, "Task Offloading in Fog Computing with Deep Reinforcement Learning: Future Research Directions Based on Security and Efficiency Enhancements," in *CLOUD COMPUTING 2024 (2024)*, Jul. 2024, p. 34. arXiv: 2407.19121.

[6] K. Peng, Y. Yang, S. Wang, P. Xiao, and V. C. M. Leung, "Reliability-Aware Proactive Offloading in Mobile Edge Computing Using Stackelberg Game Approach," *IEEE Internet of Things Journal*, vol. 11, no. 9, pp. 16 660–16 671, May 2024, ISSN: 2327-4662. DOI: 10.1109/JIOT.2024.3354700.

[7] I. A. Elgendy, W. Zhang, Y.-C. Tian, and K. Li, "Resource allocation and computation offloading with data security for mobile edge computing," *Future Generation Computer Systems*, vol. 100, pp. 531–541, Nov. 2019. DOI: 10.1016/j.future.2019.05.037.

[8] X. He, R. Jin, and H. Dai, "Peace: Privacy-Preserving and Cost-Efficient Task Offloading for Mobile-Edge Computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 3, pp. 1814–1824, Mar. 2020. DOI: 10.1109/TWC.2019.2958091.

[9] M. Bukhsh, S. Abdullah, and I. S. Bajwa, "A Decentralized Edge Computing Latency-Aware Task Management Method With High Availability for IoT Applications," *IEEE Access*, vol. 9, pp. 138 994–139 008, 2021. DOI: 10.1109/ACCESS.2021.3116717.

[10] L. Liu, H. Zhu, T. Wang, and M. Tang, "A Fast and Efficient Task Offloading Approach in Edge-Cloud Collaboration Environment," *Electronics*, vol. 13, no. 2, p. 313, Jan. 2024. DOI: 10.3390/electronics13020313.

[11] M. Guo *et al.*, "HAGP: A Heuristic Algorithm Based on Greedy Policy for Task Offloading with Reliability of MDs in MEC of the Industrial Internet," *Sensors*, vol. 21, no. 10, p. 3513, May 2021. DOI: 10.3390/s21103513.

[12] S. Meng *et al.*, "A fault-tolerant dynamic scheduling method on hierarchical mobile edge cloud computing," *Computational Intelligence*, vol. 35, no. 3, pp. 577–598, Aug. 2019. DOI: 10.1111/coin.12219.

[13] G. White and S. Clarke, "Short-Term QoS Forecasting at the Edge for Reliable Service Applications," *IEEE Transactions on Services Computing*, vol. 15, no. 2, pp. 1089–1102, Mar. 2022. DOI: 10.1109/TSC.2020.2975799.

[14] T. Dreibholz and S. Mazumdar, "Towards a lightweight task scheduling framework for cloud and edge platform," *Internet of Things*, vol. 21, no. October 2022, p. 100 651, Apr. 2023. DOI: 10.1016/j.iot.2022.100651.

[15] X. Chen, Q. Shi, L. Yang, and J. Xu, "ThriftyEdge: Resource-Efficient Edge Computing for Intelligent IoT Applications," *IEEE Network*, vol. 32, no. 1, pp. 61–65, Jan. 2018. DOI: 10.1109/MNET.2018.1700145.

[16] S. Rahman *et al.*, "Resource Management Across Edge Server in Mobile Edge Computing," *IEEE Access*, vol. 12, pp. 181 579–181 589, 2024, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2024.3503058.

[17] M. Maray, E. Mustafa, J. Shuja, and M. Bilal, "Dependent task offloading with deadline-aware scheduling in mobile edge networks," *Internet of Things*, vol. 23, p. 100 868, Oct. 2023. DOI: 10.1016/j.iot.2023.100868.

[18] Y. Zhang, X. Lan, J. Ren, and L. Cai, "Efficient Computing Resource Sharing for Mobile Edge-Cloud Computing Networks," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1227–1240, Jun. 2020, ISSN: 1063-6692. DOI: 10.1109/TNET.2020.2979807.

[19] R. Mahmud, A. N. Toosi, K. Ramamohanarao, and R. Buyya, "Context-Aware Placement of Industry 4.0 Applications in Fog Computing Environments," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 11, pp. 7004–7013, Nov. 2020. DOI: 10.1109/TII.2019.2952412.

[20] M. Alkhalaileh, R. N. Calheiros, Q. V. Nguyen, and B. Javadi, "Performance Analysis of Mobile, Edge and Cloud Computing Platforms for Distributed Applications," in *Mobile Edge Computing*, Cham: Springer International Publishing, 2021, pp. 21–45. DOI: 10.1007/978-3-030-69893-5_2.

[21] A. Alwabel, R. Walters, and G. B. Wills, "DesktopCloudSim : Simulation of Node Failures in The Cloud," in *The Sixth International Conference on Cloud Computing, GRIDs, and Virtualization CLOUD COMPUTING 2015*, Nice, France: iaria, 2015.

[22] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software - Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011. DOI: 10.1002/spe.995.