International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING

ISSN:2147-6799 www.ijisae.org Original Research Paper

Comprehensive Review and Analysis on Mobile Cloud Computing and Users Offloading using Improved Optimization Approach for Edge Computing

[1]Chiti Nigam, [2]Dr. Gajanand Sharma, [3]Dr. Ekta Menghani

Submitted: 06/06/2022 **Accepted:** 10/09/2022

Abstract-This paper gives a depth overview about the technology mobile computing. It has been discussed here how these days cloud computing has reached into mobile and has helped us to reduce the battery and storage related issues in mobiles. Apart from smart phones mobile computing has affected other areas such as homes, offices, supermarts etc. Mobile cloud computing is fast and flexible. As a result, mobile cloud computing makes it easy for developers to create and share mobile app resources with end-users. Therefore, mobile applications can be built and updated faster .Mobile cloud computing shares resources. Mobile apps that run off the cloud aren't constrained by any mobile device's processing and storage limitations. All data-intensive processes can run from the cloud. This advantage means that any mobile device with access to a network can use mobile cloud apps, regardless of the operating system. Thus, users can enjoy cloud computing with Android or OS device. Sometimes fault tolerance is an issue relating to mobile cloud computing so to provide the high services without any noise interconnected high speed networks are provided . Mobile Cloud Computing (MCC) is an emerging technology that helps us in removing the shortcomings associated with the mobile computing. There is no need to download all the software that are required by the user as MCC makes it readily available. With the help of distributed data storage methods and parallel processing the process is enhanced giving a great experience to the user. In this both the data storage and processing happens outside the device. In this era mobile computing has become a trend in IT.[Kori,et.al,2019]. The users can obtain the maximum benefit of mobile technology when it is combined with the cloud technology. Mobile cloud computing also provides access to the people who are residing in the rural areas the various mobile services such as navigation, entertainment, commerce, storage and so on. Mobile cloud computing uses integrated data. Mobile cloud computing lets users securely and quickly collect and integrate information from many sources, no matter where the data is. The architecture of mobile computer with its benefits is presented in this paper.

Keywords: *Mobile Computing, Security, Architecture, Distributed data, Parallel processing, Peer Processing.*

1 Introduction

These days' mobile devices have become an integral part of everyones life. We cannot imagine our life without mobile phones. With the high usage of mobile phones mobile computing is the new emerging field these days. **As** every application is now available in mobile phones whether it is internet banking or whether it is games or image processing, it calls for high storage demands. Here comes the role of mobile cloud computing.

We all have noticed that when many applications are installed on our mobile phones sometimes our phone becomes very slow. This occurs due to[1] limited hardware and software issues. We cannot change the hardware as it has been designed in a certain way but definitely, we can change the software of our mobiles and enhance it.

1.1 Mobile Computing

Mobile computing combines the storage and processing issues with each other. It basically eliminates the need of physical wires or cables and makes use of wireless technology. It transfers data and information by the means of wireless devices like mobiles and laptops.

There are various types of mobile computing:

- Fixed with physical cable
- Fixed without physical cable
- Fixed and mobile
- Fixed and wireless

[1,2,3] Computer Science
JECRC, Jaipur
[1]chiti.johri08@gmail.com
[2]Gajanand.sharma@jecrcu.edu.in
[3]ekta.menghani@jecrcu.edu.in

Fixed and physical cable: In this configuration, the connection is made with the help of physical links that are at a fixed position.

Fixed without physical cable: In this configuration, the wireless links are used for connecting the devices.

Mobile and Wired: In this, some devices are wired, and some are mobile. They altogether make communication with other devices.

Mobile and Wireless: In this setup, the gadgets can speak with one another regardless of their situation. They can likewise interface with any organization without the utilization of any wired gadget. Massive IoT (Internet-of-Things) applications, such as face recognition, self-driving, smart healthcare, and virtual reality (VR), have emerged in recent years as microelectronic technology and embedded computing have advanced [1], [2]. Many of these applications need a large amount of compute power and rigorous latency assurances, resulting in high energy consumption on end devices. Unfortunately, most IoT devices have limited computation and battery capacity, making such complicated applications difficult to support. Rather than developing lightweight algorithms, Mobile Edge Computing (MEC) [5] [6] is a potential direction. Unlike typical cloud computing, MEC servers are located closer to mobile consumers, resulting in a significant reduction in communication delay [7]. Ultra-dense edge networks are a promising area that has drawn a lot of research attention in order to fully leverage this advantage [8] [9]. Mobile users (IoT devices) will benefit from the edge servers' widespread and smooth computation and communication support if a significant number of connected edge servers are deployed for the target mobile network [10]–[12]. Edge servers in ultra-dense edge networks can communicate with each other over high-speed connections.On a mobile device, tasks from one application might be offloaded to several edge servers [13]. As a result, IoT users' computation-intensive and latency-critical applications can be supported effectively. One of the important technologies in the edge computing paradigm is task offloading, which determines when and how a work should be offloaded to edge servers. Offloading some subtasks to edge servers can potentially enable parallel execution at both edge servers and end devices in the case of a complex calculation task. The reliance of the sub tasks, however, might have a significant impact on this parallelism. Two subtasks, for example, can be conducted in parallel if they are independent of one another. If the output of one subtask is dependent on the output of another subtask, the two tasks must be completed in the order listed. Although end-device energy consumption can be lowered, the overall delay may not be reduced since offloaded subtasks have a round-trip communication time between end-devices and edge servers.

2 Related Works

There have been some ground-breaking works on task offloading for edge computing. For example, Yang et al. [14] introduced the Potential Game based Offloading Algorithm (PGOA) to execute distributed compute offloading based on game theory, and Chen et al. [15] offered Software Defined Task Offloading (SDTO) to reduce offloading time while preserving device battery life. However, because most of them neglect the subtle relationships among the tasks, these efforts may underutilize the edge resources and add unnecessary delay to job executions. There are some existing efforts that take task dependencies into account. Task dependencies are modelled as a serial task graph by Ning et al. [17], who offer a heuristic technique to manage resource rivalry among numerous users. Task dependencies are modelled as a tree-structured task graph by Kao et al. [13], who present a lightweight offloading technique that guarantees performance. These studies are limited to certain job contexts and cannot enable edge computing task offloading in general. In this research, we consider the tasks as directed acyclic graphs (DAGs), which are more general in describing multiple task topologies than the previous efforts. DAG has been shown in several experiments to be capable of supporting a wide range of computational operations [18].

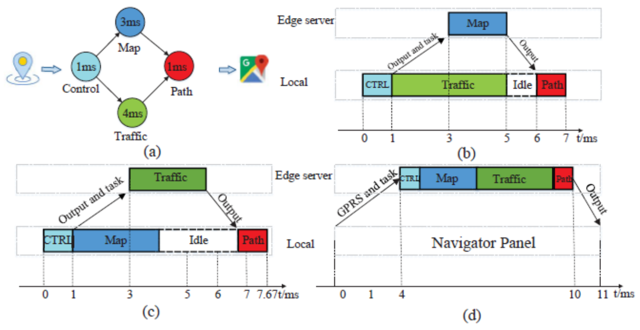

Figure 1- DAG of Navigator

Table-1. Comparison of Delay

	Transmission	Execution	Overall
Uploading Map	3 ms	6 ms	7 ms
Uploading Traffic	3 ms	6.67 ms	7.67 ms
Uploading application	5 ms	6 ms	11 ms

We present a joint optimization model for minimising task latency and energy consumption based on DAG- tasks, and then a heuristic approach that incorporates both subtask dependencies and priorities. The task dependency is ensured, as well as the the efficiency of execution between edge servers and end devices has been increased.

We show that task execution efficiency is vastly increased when compared to previous work using simulated experiments. The following are the key contributions:

• We examine task offloading with general subtask dependencies and formalise the task dependencies as directed acyclic networks (DAGs).

• To jointly optimise latency and energy usage, we develop a DAG-task offloading model and a heuristic algorithm that takes into account both subtask dependencies and priority. Task dependability is ensured, and execution efficiency is raised.

• To test the suggested algorithm's performance, we run simulation tests. The results reveal that the suggested technique significantly reduces execution delay and improves energy efficiency when compared to previous efforts.

A. Scenario of offloading

Situations for MEC offloading fall into one of three categories: (1) single-server with a single user, (2) single-server with multiple users, and (3) multi-server with numerous users. In the first scenario, Mao et al. [19] investigate the one-server MEC system with a single energy collecting device. The authors offer an online method with low complexity that is based on Lyapunov optimization. In the second case, Chen et al. [20] look at the multi-user compute offloading issue in a single-server MEC system. They first demonstrate the NP-hardness of the optimization issue before introducing a distributed offloading approach based on game theory that can reach Nash equilibrium within a limited number of iterations. In addition to the edge server, Guo et al. [21] provide two methods for cloud-MEC compute offloading via hybrid fiber-wireless networks. For the final and most complicated case, Kao et al. [13] look into multi-user compute offloading in multi-server networks. They introduce Hermes, a dynamic programming-based method, before demonstrating its time complexity and performance guarantee. With the exception of [13], none of the efforts mentioned above addressed edge server coordination, which might yield considerable advantages if properly utilized. This article discusses the multi-server and multi-user MEC scenario and takes edge server collaboration into account.

3 Methodology

Atomic task models and divisible task models are the two main genres of task model literature currently available. The atomic task model views tasks as indivisible units, which necessitates the completion of a user's job in its entirety, either locally or on edge servers. adopt a task model like this and [19], [22], and [23]. The serial task graph, in which subtasks must be completed one at a time, is the subject of the majority of current research on the divisible task model. There are two studies [17] and [24] that employ the serial task model. In [13] and [25], the tree-structured task offloading is investigated. The proposed heuristic solution in [13] focuses on a tree-structured task network and makes use of dynamic programming to optimize execution delay while remaining within the energy budget. [25] suggests Clonecloud, a technology that automatically offloads apps to the cloud. In order to establish the most effective method of offloading, our system first constructs a profile tree for each task. When it comes to the generic task dependence model, there is substantially less information accessible. The tasks are presented in [26] as a broad dependency network, although they are only brute-force solved using the ILP solver. We approach the assignments as a series of challenges rather than as prior endeavors. DAGs are a particular kind of diagram that can be used to represent a variety of operations. Numerous new research papers have been published as a result of the interest that Mobile Edge Computing (MEC) has aroused in both academia and industry. In our evaluation of the literature, we highlight three crucial ideas: the offloading circumstance, the task model, and the offloading approach.

C. Techniques for offloading

As was previously mentioned, task dependency has already been considered in a number of works. Due to task dependency's complexity, offloading solutions must consider how to handle it. [13] explores the recursive characteristics of a task graph that is tree-structured and uses dynamic programming to determine the most effective offloading strategy. Their heuristic approach may take a long time if the task graph is large, which is a pain in their side. Application offloading is the focus of Shu et al[27] .'s analysis of general subtask reliance. They create an adaptive distributed offloading strategy based on game theory that, after a limited number of iterations, reaches Nash equilibrium. However, this algorithm could only produce limited findings because of a lack of or even outdated global information. Additionally, this study ignores energy usage, a significant obstacle for edge devices. In contrast to other efforts, we describe in this research a centralized method based on subtask priorities. This technique minimizes execution latency and energy use.

We introduce the specifics of the system model in this part. A code profiler, system profiler, and decision module make up the IoT system [28]. Determining which portion of the code could be offloaded is the task of the code profiler (depending on application type and code partitioned [29]). The system profiler keeps track of variables including wireless bandwidth, upload data size, and the amount of energy used to execute or transmit each subtask. The decision module then decides whether or not to offload the subtasks. Our work primarily focuses on the decision module's design. Our decision module's major goal is to reduce application execution time while still ensuring that IoT device energy consumption is met. To be more specific, the choice to offload the work (context) to the edge servers is made to determine whether doing so will benefit IoT devices in terms of energy consumption and execution latency (Note that there is some non-offloadable code part that cannot be offloaded, e.g., user input, camera, or acquire position that needs to be executed at the IoT device). We take into account a collection of N= [1, 2,..i,..N] IoT devices, where each user has an application-level task that needs to be finished with the aid of the edge servers. There will be a variety of edge servers in the multi-access edge network or ultra-dense IoT network, described by S= [1, 2,..k,..S]. Even though there are multiple edge servers available, only one of them can be chosen by an IoT device during task execution because they only have one radio. Therefore, any IoT device I has a maximum of two offloading options to complete its subtasks: local computation or offloading to an edge server. A set of Vi= [1, 2,..j,..Vi], I N is used to represent the interdependent subtasks that make up the computation-intensive application. As shown in Fig. 1, where V is the set of subtasks, we use a directed acyclic task graph G = (V,E) to express the dependency relationships between these tasks. We presume that each subtask will be completed in an atomic manner without interruption. E is the collection of directed edges that represent the interdependence of the subtasks. For instance, a directed edge(m, n) suggests that task n depends on task m's outcome. The predicted local execution time is indicated on each node's label, and the weight of each edge indicates how much.

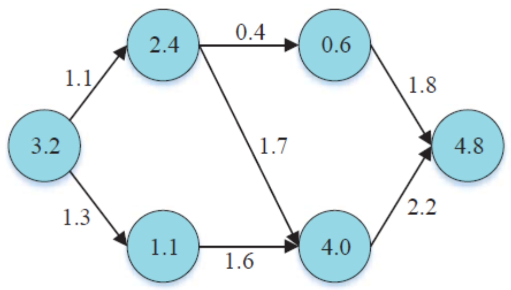

Figure 2- Example of DAG Task Scheduling

In this part, we outline an effective task offloading strategy based on the Eq. optimization issue (11). The EFO (Earliest-Finish time Offloading) technique, which depends on both the compute burden of the subtask and the dependencies between subtasks in DAGs, is the first thing we put forth for the single-user MEC system with only one edge server. With the intention of coordinating the competition of communication and computation among numerous users, we further extend the aforementioned EFO method to the multi-user MEC systems with heterogeneous servers. Additionally, we discuss a distributed computation offloading technique to carry out offloading judgments in a lightweight manner in order to increase the effectiveness of offloading decisions. The EFO Algorithm, first Scheduling subtasks to meet the task dependency criterion while minimizing latency is crucial for single-user MEC systems with a single server. There are two key issues that need to be fixed: 1) Sorting subtasks according to priorities In our work, each application is made up of a number of smaller jobs that are frequently characterized by DAGs. It's important to know how to prioritize the subtasks. Where there exist task dependency linkages, we cannot reverse the execution order. Additionally, altering the sequence in which parallel subtasks execute can have an effect on the overall latency. For instance, if the subtask Map is executed before the subtask Traffic, the computation time for the navigator will increase to 7.67ms (Note that we still keep the capability proportion of edge server and local CPU). 2) Processor selection issue: After determining the priority of each subtask, we must schedule each chosen task on its "best" processor in order to minimize the overall delay (local CPU or the edge server). As seen in Fig. 1(c), if we choose the subtask Traffic as the first one but improperly upload it to the edge server, the subtask Map will be executed locally. As a result, navigator's computation time will likewise rise to 7.67 ms.

As was previously mentioned, an IoT application is represented by a directed acyclic graph, G= (V,E). Assume that Data is a matrix of communication data, and that data a (j',j) represents the volume of information that needs to be sent from subtask j' to subtask j. Additionally, the edge (j',jcommunication)'s cost is determined by:$c_{j',j} = \begin{cases} 0 & \text{if } a_{i,j} = a_{i,j'} \\ \text{data}_{j',j}/r_{i,k}\text{ (a)} & \text{otherwise} \end{cases}$

when both subtask j and j' are scheduled on the same processor, $c_{j,j'}$ becomes zero since we assume that the intraprocessor communication cost is negligible. The average communication cost of edge (j, j') is defined by

$$\overline{c_{j,j'}} = \text{data}_{j',j}/\left(2r_{i,k}(\mathbf{a})\right)$$

In this part, we outline an effective task offloading strategy based on the Eq. optimization issue (11). The EFO (Earliest-Finish time Offloading) technique, which depends on both the compute burden of the subtask and the dependencies between subtasks in DAGs, is the first thing we put forth for the single-user MEC system with only one edge server. With the intention of coordinating the competition of communication and computation among numerous users, we further extend the aforementioned EFO method to the multi-user MEC systems with heterogeneous servers. Additionally, we discuss a distributed computation offloading technique to carry out offloading judgments in a lightweight manner in order to increase the effectiveness of offloading decisions. The EFO Algorithm, first Scheduling subtasks to meet the task dependency criterion while minimizing latency is

crucial for single-user MEC systems with a single server. There are two key issues that need to be fixed: 1) Sorting subtasks according to priorities In our work, each application is made up of a number of smaller jobs that are frequently characterized by DAGs. It's important to know how to prioritize the subtasks. Where there exist task dependency linkages, we cannot reverse the execution order. Additionally, altering the sequence in which parallel subtasks execute can have an effect on the overall latency. For instance, if the subtask Map is executed before the subtask Traffic, the computation time for the navigator will increase to 7.67ms (Note that we still keep the capability proportion of edge server and local CPU). Processor selection issue: After determining the priority of each subtask, we must schedule each chosen task on its "best" processor in order to minimize the overall delay (local CPU or the edge server). As seen in Fig, if we choose the subtask Traffic as the first one but improperly upload it to the edge server, the subtask Map will be executed locally. As a result, navigator's computation time will likewise rise to 7.67 ms. As was previously mentioned, an IoT application is represented by a directed acyclic graph, G = (V,E). Dataj ,j is the quantity of data needed to be communicated from subtask j_ to subtask j, therefore let Data be a matrix of communication data.

As a result, we expand the EFO algorithm to the multi-user case with several servers in this subsection. Since various users do not have access to one another's offloading techniques, they have limited knowledge of the wireless channel conditions and the computation load on the edge servers when they offload their computing duties. The information about the strategy comprises the servers they select, the number of subtasks they upload, and the task scheduling in edge servers. Although it hasn't been fully edited, this article has been accepted for publication in a subsequent edition of this magazine. Before the final publishing, the content may change. The scattered IoT devices can be logically controlled from a central location using the SDWN (Software Defined Wireless Network) [31]. Each IoT device uses the SDWN technology to initially upload its task-related DAGs and local processing capabilities to the controller. The SDWN controller will then choose which server (different server) and what time (task scheduling) these subtasks, which belong to various users, should be executed on.

The centralized EFO algorithm starts by listing all the possible optional offloading options. Let ai represent the only offloading option that is available to all users. We use the EFO algorithm to determine the overall computation time of each offloading scheme by treating the task graphs of the users who offload to the same server as an integrated DAG by the union of the graph. Now that we are aware of the delay for each unique offloading strategy. In the end, the offloading approach that has the least latency is chosen. Theorem 2 states that as the number of servers and users grows, so will the algorithm 2's convergence time. Then, in order to increase the effectiveness of the offloading decisions, we will talk about and construct a distributed computing offloading algorithm. It may be difficult to execute the centralized computational offloading technique in extremely dense IoT and edge networks since it would result in significant overhead as users and servers increased. Even worse, if the controller experiences a hardware breakdown, it can result in system failures. Additionally, it is challenging to develop the same standard for a variety of products because different IoT devices are typically held by different suppliers. We are motivated by these elements to create a distributed computation offloading technique. Using dispersed approaches, each IoT device can decide locally depending on the data it gathers (e.g., the channel conditions broadcasted by the edge servers). In our work, we apply a game-theoretic strategy to manage rivalry between numerous users. A potent tool for creating distributed systems with little complexity is game theory. A rational user responds to other players' actions in the preceding phases at each step of a multi-user, multi-server offloading method based on game theory and makes a locally optimal choice. All users have the ability to self-organize into the Nash equilibrium after a finite number of steps. In such a situation, no user can unilaterally alter its strategy to further reduce its delay. With the exception of user I let ai represent the computation offloading choices made by all other users. In light of the tactics used by other users, user I would like to make a locally optimum offloading decision for each of its subtasks, hence reducing its overall delay: The system finally reaches Nash equilibrium after a finite number of iterations, and we may then obtain offloading solutions for all IoT devices. We use an example from Fig. 2 to try to demonstrate our DEFO technique. In the network, there are two edge servers and three IoT devices. Every device calls the EFO algorithm during the first iteration to compute AFT in accordance with a different target server for offloading.It should be noted that every device at this point expects that the wireless channel is free of interference and that there aren't any other subtasks running on the edge server. The controller will then decide that user 3 should offload their subtask to server 1 because it results in the greatest delay reduction relative to the local execution time (line 10). User3 applies the EFO method in lines 14–16 and transfers tasks 2 and 4 to server1. The IoT devices will then be broadcasted by server server1 with the scheduling results and its wireless channel rates ri,k(a), enabling the users to make additional decisions in the subsequent iteration.

4 Results

In this section, we assess the offloading algorithm's performance that was suggested in Section IV. Several mobile end devices are randomly positioned in our simulation scenario's area that is covered by extremely dense edge networks, with the edge servers connected to one another over high-speed fiber networks. The edge servers' computing and communication skills are diverse. A single DAG-task is randomly allocated to each device. The computational and communication burden of each DAG task is distributed at random. The end-to-end performance of task offloading is significantly influenced by both the number of servers and the number of end devices. We concentrate on the impact of the quantity of servers and end devices as a result. Performance of the offloading algorithm as a function of the number of servers The first thing is how many servers there are.

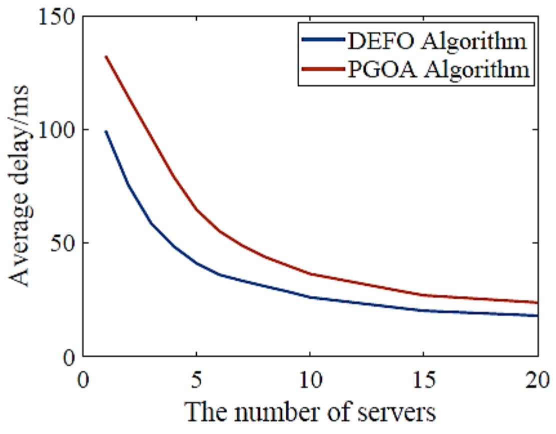

Figure. 3- Average Delay for Different Network Scales

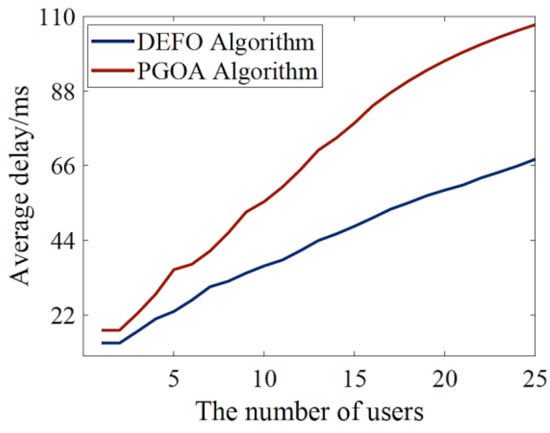

Figure. 4- Average Delay for Different User Scales

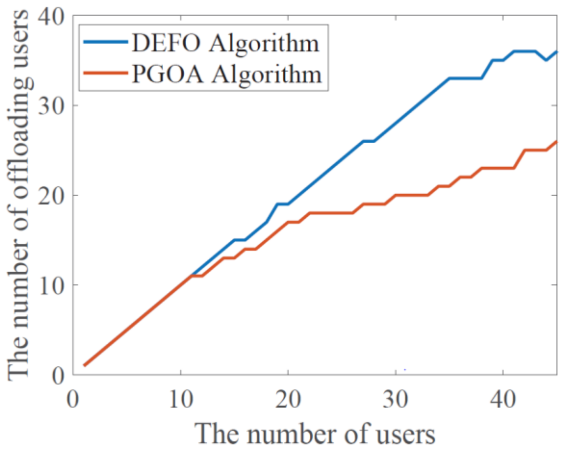

Figure 5- Offloading Users for Different User Scales

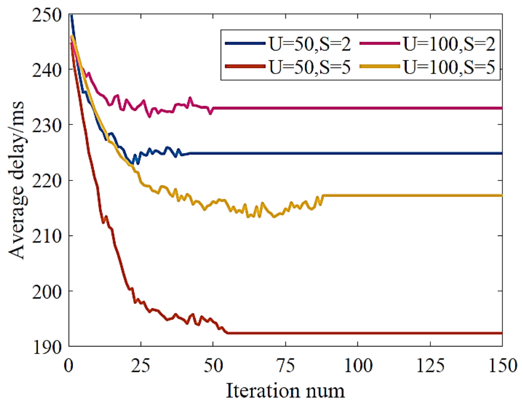

Figure 6- The convergence of DEFO algorithm

We fix things. Increase the number of devices to 100 and the number of edge servers from 1 to 50 in order to monitor mobile device latency and energy usage. As seen in Figure 3. The average latency of mobile devices decreases as the number of edge servers rises. Subtasks can be more parallelized because more servers give mobile users more offloading possibilities. Additionally, our method is contrasted with two already-in-use offloading schemes: the single-server DAG-task offloading algorithm and the PGOA presented in . The DEFO is unable to take advantage of DAG parallelism because it views the tasks performed by mobile users as being inseparable. The DEFO is a modification to our offloading technique that prevents it from using edge-server networks by requiring all subtasks on a single device to be offloaded to the same edge server (if they choose to be offloaded). Additionally, we found that as the number of servers increases, the latency difference between DEFO and our offloading technique widens, suggesting that our offloading algorithm is better able to take advantage of parallel processing. Fig. Figure 4 displays the algorithms' energy use. Because this method offloads an entire operation to the server for execution, which decreases local processing but results in greater latency, it should be noted that we are not comparing PGOA's energy consumption. Combining Figures 1 and 2 Figures 2 and 3 Figures 1 and 2. We can see that our algorithm responds more quickly while using less energy. Performance of the offloading method on different mobile platforms. Next, it is looked into how many mobile devices there are. The results are shown in Figures 4 and 5. The number of mobile devices is increased from one to one hundred, but the number of edge servers remains constant at 20. We uniformized the DAG-task for every device in order to isolate the impact of the number of devices. As seen in Figure 4, Our offloading approach offers the lowest latency performance among the three methods. It's important to note that the PGOA and DEFO average latency do not change when there are fewer than 20 devices. There is no resource contention among them because there are more servers than devices, which is the cause of this. These two techniques can only partially utilize several servers because a device can only offload to one server.Contrarily, our approach lacks this restriction, enabling it to efficiently utilize edge resources. It is important to note that the latency difference between PGOA and DEFO expands as the number of devices increases. This conclusion explains the benefits of work splitting. As the number of devices increases, there are greater advantages since more devices increase parallelism. Fig. Figure 5 displays the energy utilization of several algorithms. Similar to the earlier finding, our method uses less energy while providing superior latency performance.

5. Conclusions

In this paper, we study task offloading in ultra-dense edge networks. We first develop the DAG-task offloading problem, which optimizes task delay and energy consumption, and then employ directed acyclic graphs (DAGs) to describe the general task dependency of mobile apps. In view of the problem's complexity, we then propose a heuristic priority-based DAG-task offloading approach. The method effectively offloads jobs while preserving task dependency structures. The evaluation results show that our approach can decrease latency and energy consumption in extremely dense edge networks. Our performance gain in this job is primarily due toend-to-end parallelism between end devices and edge servers is maximised. In the future, we'll concentrate on assessing the degree of parallelism among subtasks, confirming the performance of our algorithm under various levels of parallelism, and explicitly implementing parallelism into our offloading method.

REFERENCES

1. Badea, Gheorghe, Raluca-Andreea Felseghi, Mihai Varlam, Constantin Filote, Mihai Culcer, Mariana Iliescu, and Maria Simona Răboacă. "Design and simulation of romanian solar energy charging station for electric vehicles." Energies 12, no. 1, 2019.
2. Singh, Dushyant, and Baldev Singh. "Secure Chess-Based Data Exchange and User Validation." *Journal of Cases on Information Technology (JCIT)* 24.4 (2022): 1-10.
3. Singh, Dushyant. "A Review on Deep Learning Models." *Integrated Emerging Methods of Artificial Intelligence & Cloud Computing* (2022): 223-229.
4. Rai, S. K. ., Rana, D. P. ., & Kashif, D. M. . (2022). Hotel Personnel Retention In Uttar Pradesh: A Study of HYATT Hotels. International Journal of New Practices in Management and Engineering, 11(01), 47–52. https://doi.org/10.17762/ijnpme.v11i01.173
5. Jadaun, A., Alaria, S.K. and Saini, Y. 2021. Comparative Study and Design Light Weight Data Security System for Secure Data Transmission in Internet of Things. *International Journal on Recent and Innovation Trends in Computing and Communication*. 9, 3 (Mar. 2021), 28–32. DOI:https://doi.org/10.17762/ijritcc.v9i3.5476.
6. Ashish, Vijay Kumar, Satish Kumar Alaria, Vivesk Sharma "Design Simulation and Assessment of Prediction of Mortality in Intensive Care Unit Using Intelligent Algorithms", *Mathematical Statistician and Engineering Applications*. 71, 2 (May 2022), 355–367. DOI:https://doi.org/10.17762/msea.v71i2.9
7. Satish Kumar Alaria and Abha Jadaun, "Design and Performance Assessment of Light Weight Data Security System for Secure Data Transmission in IoT", Journal of Network Security, Vol.: 9, Issue: 1, (2021) PP: 29-41.
8. Khandelwal, Ravi, Manish Kumar Mukhija, and Satish Kumar Alaria. "Numerical Simulation and Performance Assessment of Improved Particle Swarm Optimization Based Request Scheduling in Edge Computing for IOT Applications." *New Arch-International Journal Of Contemporary Architecture* 8, no. 2 (2021): 155-169.
9. S. Hu and G. Li, "Dynamic Request Scheduling Optimization in Mobile Edge Computing for IoT Applications," in IEEE Internet of Things Journal, vol. 7, no. 2, pp. 1426-1437, Feb. 2020, doi: 10.1109/JIOT.2019.2955311.
10. Sehirli, E., & Alesmaeil, A. (2022). Detecting Face-Touch Hand Moves Using Smartwatch Inertial Sensors and Convolutional Neural Networks. International Journal of Intelligent Systems and Applications in Engineering, 10(1), 122–128. https://doi.org/10.18201/ijisae.2022.275
11. M. Smith, A. Maiti, A. D. Maxwell, and A. A. Kist, "Object detection resource usage within a remote real-time video stream," in Online Engineering & Internet of Things, Cham, Switzerland: Springer, 2018, pp. 266–277.
12. P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," IEEE Commun. Surveys Tut., vol. 19, no. 3, pp. 1628–1656, 3rd Quart., 2017.
13. Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," IEEE Commun. Surveys Tuts., vol. 19, no. 4, pp. 2322–2358, 4th Quart., 2017.
14. H. A. Alameddine, S. Sharafeddine, S. Sebbah, S. Ayoubi, and C. Assi, "Dynamic task offloading and scheduling for low-latency IoT services in multi-access edge computing," IEEE J. Sel. Areas Commun., vol. 37, no. 3, pp. 668–682, Mar. 2019.
15. M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," IEEE J. Sel. Areas Commun., vol. 36, no. 3, pp. 587–597, Mar. 2018.
16. X. Lyu, H. Tian, C. Sengul, and P. Zhang, "Multiuser joint task offloading and resources optimization in proximate clouds," IEEE Trans. Veh. Technol., vol. 66, no. 4, pp. 3435–3447, Apr. 2017.
17. Q. Wang, S. Guo, J. Liu, and Y. Yang, "Energy-efficient computation offloading and resource allocation for delay-sensitive mobile edge computing," Sustain. Comput. Informat. Syst., vol. 21, pp. 154–164, Mar. 2019.

18. T. X. Tran and D. Pompili, “Joint task offloading and resource allocation for multi-server mobile-edge computing networks,” IEEE Trans. Veh. Technol., vol. 68, no. 1, pp. 856–868, Jan. 2019.
19. M. Shojafar, N. Cordeschi, and E. Baccarelli, “Energy-efficient adaptive resource management for real-time vehicular cloud services,” IEEE Trans. Cloud Comput., vol. 7, no. 1, pp. 196–209, Jan.–Mar. 2019.
20. S. M. R. Islam, N. Avazov, O. A. Dobre, and K.-S. Kwak, “Powerdomain non-orthogonal multiple access (NOMA) in 5G systems: Potentials and challenges,” IEEE Commun. Surveys Tuts., vol. 19, no. 2, pp. 721–742, 2nd Quart., 2017.
21. M. Kamel, W. Hamouda, and A. Youssef, “Ultra-dense networks: A survey,” IEEE Commun. Surveys Tuts., vol. 18, no. 4, pp. 2522–2545, 4th Quart., 2016.
22. Gill, D. R. . (2022). A Study of Framework of Behavioural Driven Development: Methodologies, Advantages, and Challenges. International Journal on Future Revolution in Computer Science &Amp; Communication Engineering, 8(2), 09–12. https://doi.org/10.17762/ijfrcsce.v8i2.2068
23. D. López-Pérez, M. Ding, H. Claussen, and A. H. Jafari, “Towards 1 Gbps/UE in cellular systems: Understanding ultra-dense small cell deployments,” IEEE Commun. Surveys Tuts., vol. 17, no. 4, pp. 2078–2101, 4th Quart., 2015.
24. B. Yu, L. Pu, Q. Xie, and J. Xu, “Energy efficient scheduling for IoT applications with offloading, user association and BS sleeping in ultra dense networks,” in Proc. 16th Int. Symp. Model. Optim. Mobile Ad Hoc Wireless Netw. (WiOpt), Shanghai, China, 2018, pp. 1–6.
25. C. Ma, F. Liu, Z. Zeng, and S. Zhao, “An energy-efficient user association scheme based on robust optimization in ultra-dense networks,” in Proc. IEEE/CIC Int. Conf. Commun. China (ICCC Workshops), Beijing, China, 2018, pp. 222–226.
26. X. Chen, L. Jiao, W. Li, and X. Fu, “Efficient multi-user computation offloading for mobile-edge cloud computing,” IEEE/ACM Trans. Netw., vol. 24, no. 5, pp. 2795–2808, Oct. 2016.
27. T. V. Do, N. H. Do, H. T. Nguyen, C. Rotter, A. Hegyi, and P. Hegyi, “Comparison of scheduling algorithms for multiple mobile computing edge clouds,” Simulat. Model. Pract. Theory, vol. 93, pp. 104–118, May 2019.
28. L. Gu, J. Cai, D. Zeng, Y. Zhang, H. Jin, and W. Dai, “Energy efficient task allocation and energy scheduling in green energy powered edge computing,” Future Gener. Comput. Syst., vol. 95, pp. 89–99, Jun. 2019.
29. Y. Jie, X. Tang, K.-K. R. Choo, S. Su, M. Li, and C. Guo, “Online task scheduling for edge computing based on repeated Stackelberg game,” J. Parallel Distrib. Comput., vol. 122, pp. 159–172, Dec. 2018.
A. Kiani and N. Ansari, “Toward hierarchical mobile edge computing: An auction-based profit maximization approach,” IEEE Internet Things J., vol. 4, no. 6, pp. 2082–2091, Dec. 2017.
30. K. Lin, S. Pankaj, and D. Wang, “Task offloading and resource allocation for edge-of-things computing on smart healthcare systems,” Comput. Elect. Eng., vol. 72, pp. 348–360, Nov. 2018.
31. T. Wang, G. Zhang, A. Liu, M. Z. A. Bhuiyan, and Q. Jin, “A secure IoT service architecture with an efficient balance dynamics based on cloud and edge computing,” IEEE Internet Things J., vol. 6, no. 3, pp. 4831–4843, Jun. 2019.
32. T. Bahreini, H. Badri, and D. Grosu, “An envy-free auction mechanism for resource allocation in edge computing systems,” in Proc. IEEE/ACM Symp. Edge Comput. (SEC), Seattle, WA, USA, 2018, pp. 313–322.
33. S. Misra and N. Saha, “Detour: Dynamic task offloading in softwaredefined fog for IoT applications,” IEEE J. Sel. Areas Commun., vol. 37, no. 5, pp. 1159–1166, May 2019.
34. Chiba, Z., El Kasmi Alaoui, M. S., Abghour, N., & Moussaid, K. (2022). Automatic Building of a Powerful IDS for The Cloud Based on Deep Neural Network by Using a Novel Combination of Simulated Annealing Algorithm and Improved Self- Adaptive Genetic Algorithm. International Journal of Communication Networks and Information Security (IJCNIS), 14(1). https://doi.org/10.17762/ijcnis.v14i1.5264
35. H. Guo, J. Liu, and J. Zhang, “Computation offloading for multi-access mobile edge computing in ultra-dense networks,” IEEE Commun. Mag., vol. 56, no. 8, pp. 14–19, Aug. 2018.
36. H. Guo, J. Zhang, J. Liu, H. Zhang, and W. Sun, “Energy-efficient task offloading and transmit power allocation for ultra-dense edge computing,” in Proc. IEEE Global Commun. Conf. (GLOBECOM), 2018, pp. 1–6.
37. S. Jeong, O. Simeone, and J. Kang, “Mobile edge computing via a UAVmounted cloudlet: Optimization of bit allocation and path planning,” IEEE Trans. Veh. Technol., vol. 67, no. 3, pp. 2049–2063, Mar. 2018.
38. Y. Nakamura, T. Mizumoto, H. Suwa, Y. Arakawa, H. Yamaguchi, and K. Yasumoto, “In-situ resource provisioning with adaptive scale-out for regional IoT services,” in Proc. IEEE/ACM Symp. Edge Comput. (SEC), Seattle, WA, USA, 2018, pp. 203–213.
39. M. Kumar and C. Guria, “The elitist non-dominated sorting genetic algorithm with inheritance (i-NSGA-II) and its jumping gene adaptations for multi-objective optimization,” Inf. Sci., vols. 382–383, pp. 15–37, Mar. 2017.
40. Singh, Pooja, Manish Kumar Mukhija, and Satish Kumar Alaria. "An Approach for Cloud Security Using TPA-and Role-Based Hybrid Concept." In *Proceedings of Third International Conference on Computing, Communications, and Cyber-Security*, pp. 153-162. Springer, Singapore, 2023.
41. S. K. A. S. D. “Reducing the Packets Loss Using New MAC Protocol”. *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 1, no. 9, Sept. 2013, pp. 747-51, doi:10.17762/ijritcc.v1i9.2856.
42. Mishra, P. ., S. K. . Alaria, and P. . Dangi. “Design and Comparison of LEACH and Improved Centralized LEACH in Wireless Sensor Network”. *International Journal on Recent and Innovation Trends in Computing*

and Communication, vol. 9, no. 5, May 2021, pp. 34-39, doi:10.17762/ijritcc.v9i5.5478