Task-Aware Load Balancing in Mobile Cloud Computing using Cloudlets

Joseph Arockia Mary[1,*] & A. Aloysius[2]

[1]Department of Computer Applications, Holy Cross College,
Teppakulam, Trichy-620 002, Tamil Nadu, India
(Affiliated to Bharathidasan University, Palkalai Perur,
Tiruchirappalli-620 024, Tamil Nadu, India)
[2]Department of Computer Science, St. Joseph's College,
Trichy-620 002, Tamil Nadu, India.
(Affiliated to Bharathidasan University, Palkalai Perur,
Tiruchirappalli-620 024, Tamil Nadu, India)

*E-mail: arockiamarytcs@hcctrichy.ac.in

Abstract. In the current era, almost every person uses a mobile device for tasks that range from basic phone calls to advanced computations. To increase the QoE of the mobile user, tasks may be offloaded to a cloudlet because of the lack of a large battery, large memory, and sufficient processing power. Cloudlets can be heavily loaded with tasks when they are in a heavily populated area, where tasks are overflowing. Meanwhile, other cloudlets are lightly loaded and their resources are often idle when they are in a moderately populated area, getting fewer tasks. When tasks are queued up for a long time in a heavily loaded cloudlet, the response time and dropout rate of tasks increase. These two parameters deteriorate further when a big task is waiting for a long time. Such big tasks waiting for a long time can be migrated to other cloudlets to utilize the idle resources available in lightly loaded cloudlets, which may provide better QoE to the user. This article uses the Firefly optimization algorithm for choosing which tasks to migrate to which cloudlet for load balancing based on task details such as total waiting task size, task waiting time, and total virtual machine size. The proposed method, called TALBMCC (Task-Aware Load Balancing in Mobile Cloud Computing), also increases the number of tasks executed by cloudlets and reduces the execution time of tasks and the power consumption of mobile devices.

Keywords: *big data; cloudlet; dropout rate; firefly optimization algorithm; latency; load balancing; mobile cloud computing; response time; resource scheduling.*

1 Introduction

Mobile devices such as smartphones, smartwatches, and Google Glass may process large computations that exceed their capacity. While these devices process these large computations, the QoE of the user is affected, such as the execution time of the task and the power consumption of the mobile device. These parameters increase, causing the devices to become unusable as the processing,

Received October 18th, 2025, Revised May 22nd, 2026, Accepted for publication June 18th, 2026.

battery, and memory capacity of these devices is insufficient. To overcome these difficulties, tasks are offloaded to the cloud, where they are executed and the result is sent back to the user. This method improves the QoE. To further improve the QoE of the user, tasks can be offloaded to a nearby cloudlet [1] instead of a distant cloud, which consumes more network resources and several hops. Cloudlets are small data centers situated near the access point and execute tasks more easily and quickly. This strategy allows to provide an optimal online gaming experience, deliver high-quality video, enhance digital financial services, protect patient data, etc. All these tasks and their data are offloaded to cloudlets for execution. However, when too many tasks come for execution, a cloudlet can become overloaded, causing more tasks to be waiting. Thus, the waiting time of the tasks and the execution time of the tasks increase. To reduce this phenomenon, tasks can be migrated to other cloudlets, whose VMs are idle and who have a small queue of waiting tasks. The tasks are executed in nearby cloudlets to balance the load among the cloudlets. For example, in real-time applications, the load balancer uses cookies to count the number of people who are accessing websites; if the count exceeds the capacity of the cloudlet, the task is offloaded to another cloudlet. [2]

The reason some cloudlets are overloaded with tasks and data is that some of them are located in more crowded regions. Mobile data is increased by up to 78% by 2021 due to video content, as illustrated in Sinky et al. [3]. In heavily loaded cloudlets, also called *maximum cloudlets*, several tasks are waiting in line. Other cloudlets are situated in less crowded regions and obtain a lower number of tasks for execution. Therefore, their resources, such as VMs and hard disks, are idle a lot of the time. To compensate for the task load of a particular cloudlet by using idle cloudlets and thus improve the QoE of the users, several researchers have proposed different load balancing algorithms.

The method proposed in this paper ensures that all nearby cloudlets are never idle. All the resources are utilized fully without any idle time, even for a single second. To use all the available resources of the cloudlets optimally, this method uses the Firefly algorithm [4]. The Firefly algorithm is a swarm-based, meta-heuristic optimization algorithm for solving continuous problems. It is suitable for data in different groups and clusters.

The proposed method uses different groups of data from different cloudlets to find global minimum and global maximum values among the cloudlets. The groups of data it uses are: the largest waiting task in each cloudlet; the waiting time; the total waiting task size; and the total VM size of every nearby cloudlet. These are sent to a powerful central cloudlet. From these groups of data from different cloudlets, the central cloudlet chooses the largest task with the longest waiting time from each cloudlet and sends them to cloudlets where the task load

is lowest. Thus, the task waiting time and execution time are reduced. The method also reduces the dropout rate and response time and increases the number of tasks executed by cloudlets.

In this article, Section 2 describes the different algorithms proposed by other researchers to balance the load in cloudlets and new techniques and their drawbacks. Section 3 explains the proposed method using the Firefly algorithm to allow to utilize cloudlet resources optimally. Section 4 illustrates the comparison results of the proposed method with similar methods developed by other researchers. Finally, this article ends with the conclusion and future work in Section 5.

2 Literature Review

Various load balancing algorithms have been proposed to utilize cloudlets and their resources more efficiently to execute offloaded tasks from mobile devices. Cloudlets near the access point are referenced in different ways based on their names. This article discusses some of the load balancing algorithms in mobile cloud computing (MCC). In Lai et al. [5], the fairness-oriented task offloading scheme uses the FairEdge method to integrate the balls-and-bins theory with the fairness index to balance the load in mobile cloudlets. A cloudlet calculates the fairness index and shares the load information with two nearby mobile cloudlets. The method was evaluated with two real-world datasets, Mobiclique and Haggle. The drawback of this scheme is that only two nearby cloudlets are considered, so tasks may still queue up in the cloudlets. The cloudlets and their resources are not optimally utilized.

Herbert Raj et al. [6] developed a load balancing method that arranges tasks in decreasing order using the heap sort method. Bin Packing's First Fit Decreasing (FFD) algorithm is used for load balancing among the resources. The whole task is allocated to a server of equal capacity. This method allocates tasks to the minimum number of processors, for which it uses four heuristic methods, namely, First Fit, First Fit Decreasing, Best Fit, and Best Fit Decreasing. This ensures reduced waste of computational space. JongBeom Lim & DaeWon Lee [7] developed a graph coloring-based implementation that uses a genetic algorithm for load balancing edge servers, using three colors. The edge servers are depicted as vertices, while mobile devices are depicted as edges. When a task is allocated to an edge server, then the corresponding vertex is colored, provided that a nearby vertex is not colored with the same color. If none of the three colors is available, the task is allocated to the cloud only if the task's waiting time is lower than the forwarding time. However, this algorithm does not forecast the status of the edge server and the mobility of the devices is not considered.

Ranapana & Jayasena [8] introduced a new architecture for load balancing, an intermediate node, between the access point and the edge servers. This intermediate node knows the status of the edge servers. If any task is to be executed, the node finds the edge server with a limited load for offloading. The drawback of this architecture is that the intermediate node is an extra device that increases the cost of task execution and network resource consumption. The Accelerated Cuckoo Search (ACS) algorithm developed by Arun & Prabhu [9] is used to find cloudlets with an idle VM. This algorithm reduces makespan and CPU utilization. If there is no free cloudlet, the task is allocated to a random VM using the ACS algorithm. This algorithm finds a suitable VM where the full task can be executed. However, in this algorithm, tasks are not split and preempted, so there is a waste of idle VM capacity because tasks are only allocated to equal-sized VMs.

The time-balance load management method by Hussein [10] offloads tasks from mobile devices to the cloud using CPU index and CPU utilization. The Throttled algorithm checks the VM and queue status before allocating requests and supports task migration between VMs. A centralized data center controller performs VM load balancing. However, since the decisions are made by a distant cloud server rather than a cloudlet, the communication overhead increases the response time. In Lee et al. [11], a hybrid cloud architecture is proposed for healthcare task execution using private, public, and edge clouds. When a user is within the range of an edge cloud, the healthcare task is offloaded to it. As the user moves to another edge cloud, the task is migrated accordingly. The edge cloud is removed after completing its final task. However, frequent creation and deletion of edge clouds consume additional cost, time, and resources. The load balancing algorithm in Kanbar & Faraj [12], which combines Support Vector Machine (SVM) and Ant Colony Optimization (ACO), classifies tasks as sensitive or non-sensitive. Sensitive tasks are executed in a private cloud, while non-sensitive tasks run in a public cloud. Tasks are stored with energy and execution-time details, and VMs are grouped into primary, secondary, and recovery repositories. Although the algorithm reduces latency, execution time, and bandwidth consumption, it does not support VM migration.

2.1 Motivation and Challenges

The literature discussed above has left the challenges listed below, which motivated us to develop a method for load balancing in multiple cloudlets.

1. The decision to migrate a task to an available cloudlet, where one or two VMs are idle, is made by either a cloud server or a new resource is introduced that increases the network resource consumption and execution time of the task.

2. The load balancing algorithm does not consider the allocation of the task based on the available VM size in the cloudlet. Simply, it checks the entire cloudlet.
3. The waiting time and mobility of the devices are not considered in MCC.

This article addresses these challenges and focuses on the decision taken by the existing cloudlet and migrating only large tasks with a long waiting time based on the size of the available VMs in cloudlets. The main objective of this proposed load-balancing algorithm is to improve the QoS by reducing the dropout rate and response time of the cloudlets. The following techniques are applied to improve the QoS:

1. Using the existing cloudlet as a decision maker to select the task for migration from a heavily loaded cloudlet and to select a less loaded cloudlet.
2. Giving priority to large tasks and tasks with big data.
3. Considering the task's waiting time in selecting the task for migration.
4. Specifying the total number of cloudlets used by the offloaded task.

3 Proposed Method

Scheduling tasks across multiple cloudlets in order to use the resources of VMs and memory more optimally can improve the QoE of users and the performance of cloudlets, especially in the mobile world. This method uses the Firefly optimization algorithm, which finds the global minimum and maximum from different groups of data. It is suitable for load balancing among cloudlets because it is a multimodal optimization problem. The proposed method uses the Firefly algorithm for constrained optimization problems to improve cloudlet performance.

The proposed load balancing method (TALBMCC) uses the status of the tasks in each cloudlet to balance the loads among nearby cloudlets. The largest existing cloudlet is given the controller position to schedule and balance the tasks among multiple cloudlets. The controller's job is to find cloudlets that are heavily loaded with tasks and cloudlets with fewer tasks at a certain time. Thus, the tasks from a cloudlet that has a long queue of waiting tasks can be migrated to other cloudlets with fewer tasks in queue. An algorithm is needed to find tasks that are suitable for migration to least loaded cloudlet, also called the *minimum cloudlet*, without affecting the QoS of the user. To compare multiple parameters of the task and the cloudlets in one step, the Firefly optimization algorithm is used in the proposed method, with slight modifications based on the nature of the problem. The following section explains the basics of the Firefly algorithm.

3.1 Firefly Algorithm

The Firefly algorithm [13] is a swarm intelligence algorithm inspired by the behavior of fireflies. Fireflies have flashing lights in their bodies with which they attract each other depending on the brightness of the flashing. If the brightness is high because of the smaller distance between two fireflies, the attraction is greater. If the flashing is dim, it means that the distance between two fireflies is greater, and attraction is smaller. This nature of firefly attraction is expressed in the equations that make up the Firefly algorithm. Eq. (1) represents the amount of attraction between two fireflies:

$$\beta = \beta_0 e^{-\gamma r} \tag{1}$$

In Equation 1, β_0 is the attractiveness between the fireflies when the distance between the fireflies is zero. In $e^{-\gamma r}$, γ is the absorption coefficient and *r* is the distance between the fireflies. The attractiveness is high when $\gamma = 0$; then $\beta = \beta_0$. When γ = ∞ and the distance *r* is unknown, then the attractiveness $\beta = 0$, and there is no attraction because the outcome is zero. The position of the firefly in the next iteration, t + 1, is given by Eq. (2):

$$x_i^{t+1} = x_i^t + \beta_0 e^{-\gamma r_{ij}} (x_j^t - x_i^t) + \alpha \xi_i^t \tag{2}$$

In Eq. (2), the position of the firefly is defined relative to the position of other fireflies, where x_i^{t+1} denotes the position of firefly *i* at the next iteration *t* + 1; x_i^t represents the position of the firefly at the previous iteration *t*; $\beta_0 e^{-\gamma r_{ij}} (x_j^t - x_i^t)$ denotes the attraction between the *j*-th firefly and the *i*-th firefly. This attraction is calculated by finding the distance between the *i*-th and *j*-th firefly. Finding this distance is done by Eq. (3):

$$r_{ij} = \sqrt{\sum_{k=1}^{D} (x_{i,k} - x_{j,k})^2} \tag{3}$$

$x_{i,k}$ is the *k*-th component of the *i*-th firefly, and similarly, $x_{j,k}$ denotes the *k*-th component of the *j*-th firefly. These equations to calculate the attraction between two fireflies are used to balance the load in multiple cloudlets in the next subsection. The definitions for various symbols used in the algorithm are given in Table 1.

Table 1 List of notations.

Symbols	Definitions
β_0	Attractiveness between two fireflies
Γ	Absorption coefficient
r	Distance between the two fireflies
x_i^{t+1}	Position of the firefly *i* at the next iteration *t* + 1
x_i^t	Position of firefly *i* at the previous iteration *t*
$\beta_0 e^{-\gamma r_{ij}}(x_j^t - x_i^t)$	Attraction between the *j*-th firefly and the *i*-th firefly
$x_{i,k}$	*k*-th component of the *i*-th firefly
$x_{j,k}$	*k*-th component of the *j*-th firefly
dr	Dropout rate
rt	Response time
ts	Task size
n	Number of cloudlets
p	Cloudlets and their details
load_bal (n,p).	Two-dimensional array
ξ	Total number of fireflies found and total number of cloudlets
lb	Lowest ranking firefly
ub	Topmost firefly
α	Random value
r_{ij}	The distance between the *i*-th and *j*-th fireflies

3.2 Load Balancing using Firefly Optimization Algorithm

The TALBMCC method uses the Firefly algorithm for scheduling the tasks in multiple cloudlets equally to balance the loads. The objective of this method is to reduce the cloudlet dropout rate and their response time. The objective function is given in Eq. (4):

$$\text{Minimize } f(x) = \sum_{c=1}^{n} dr + rt \quad \text{with constraint } ts > 65536 \tag{4}$$

In this objective function, *c* represents the number of cloudlets *n*; *dr* is the cloudlet dropout rate; and *rt* is the cloudlet response time. To achieve load balance in the *n* cloudlets, the task size *ts* must be greater than 65,536 bytes to be selected for VM migration.

TALBMCC stores the task details in a two-dimensional array called load_bal(*n*,p). This cloudlet, which is bigger than the other cloudlets, is chosen as the controller cloudlet based on its memory and processor power. To this controller cloudlet, nearby cloudlets in four directions report their status. The controller cloudlet uses the load_bal(*n*,p) array to store the details of the other cloudlets, as shown in Table 2.

Table 2 The controller cloudlet stores the details of nearby cloudlets in load_bal(n,p).

	A	**B**	**c**	**γ**
Cloudlets/Parameters	Largest task waiting in a cloudlet (bytes)	Remaining waiting tasks in the same cloudlet (Bytes)	Total VM size (mips)	Waiting time of that largest task in first column (milliSec)
Cloudlet 1	60,000 B	55,000 B	1,000	3 ms
Cloudlet 2	10,000 B	8,800 B	2,000	1 ms
Cloudlet 3	80,000 B	165,000 B	2,500	15 ms
Cloudlet 4	90,000 B	77,000 B	2,500	3 ms
Cloudlet 5	5000 B	0	2,500	1 ms

The TALBMCC algorithm is used by the controller cloudlet, as shown in Algorithm 1. The algorithm takes task and cloudlet details as input and identifies the largest waiting task, its source cloudlet, and the target cloudlet for migration. These details are stored in the load_bal(*n*,p) array, where the four columns represent the largest waiting task (>65,536 B), remaining tasks, total VM size, and waiting time. Updated every second, the controller cloudlet selects the largest and longest-waiting task and migrates it to a cloudlet with fewer tasks, shorter waiting time, or an idle VM.

Algorithm 1. Task-Aware Load Balancing in Mobile Cloud Computing using Cloudlet

Input: load_bal[n,p])

Output: Two cloudlets for VM migration

1. TALBMCC (load_bal[*n*,p])
2. Initialize the parameter *n* = number of cloudlets
3. Initialize the parameter p = cloudlets and their details
4. Let p(1) = Largest waiting task in a cloudlet
5. Let p(2) = Remaining waiting tasks in the same cloudlet
6. Let p(3) = Total VM size of a cloudlet
7. Let p(4) = Waiting time of that largest task
8. Let *ub* = upper bound, *lb* = lower bound
9. Set all the parameters p1, p2, p3, p4, γ = 0.1, $\beta_0 = 1$, $\alpha = 1$, *ub*, *lb* = *n*, $\xi = ub\text{-}lb$
10. Set iteration variable *it* = 0
11. Set time counter ii-max = 86400 sec and *it* = 1
12. Fill the array load-bal[*n*,p] from *n*
13. While (*it*< maximum number of iterations)
14. for(*i* = 1 to *n*) do
15. for(*j* = I + 1 to *n*) do

16. for(k = 1, k <=p,*k*++)
17. Calculate the distance between the two cloudlets $r = ||x_j^t - x_i^t||2$
18. r = *r*+*rij*
19. end for
20. Calculate the attractiveness between the two cloudlets $\beta = \beta_0 e^{-\gamma r}$
21. Calculate the status of the *i*-th cloudlet f(x_i) = $x_i^{t+1} = \beta_0 e^{-\gamma r_{ij}}(x_j^t - x_i^t) + \alpha\xi_i^t$
22. Calculate the status of the *j*-th cloudlet f(x_i) = $x_j^{t+1} = \beta_0 e^{-\gamma r_{ij}}(x_j^t - x_i^t) + \alpha\xi_i^t$
23. if (f(x_i) < f(x_j)) then
24. min = f(x_i), max = f(x_j)
25. else
26. min = f(x_j), max = f(x_i)
27. end if
28. end for
29. end for
30. end while
31. Move the task for execution from max to min
32. Draw the results
33. Repeat the procedure for the next second = *it* = *it* + 1

The controller cloudlet determines two results, i.e., the statuses of the two cloudlets found by Algorithm 1. Based on cloudlet status, the maximum cloudlet has the largest waiting task and longest waiting time, while the minimum cloudlet has an idle VM or fewer tasks with shorter waiting time. In the TALBMCC algorithm, Cloudlet 3 is identified as the maximum cloudlet and Cloudlet 5 as the minimum cloudlet. Hence, the task is migrated from Cloudlet 3 to Cloudlet 5 for execution.

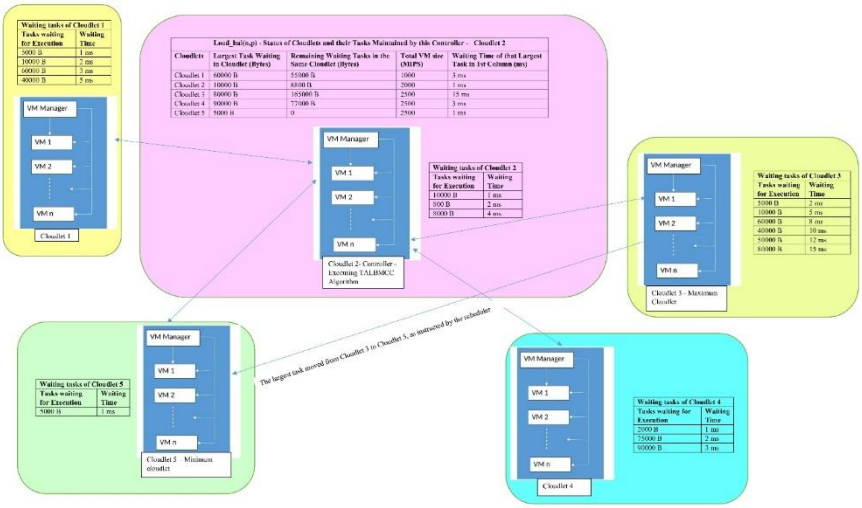

Figure 1 The controller cloudlet and other cloudlets in the TALBMCC algorithm.

TALBMCC finds the maximum cloudlet and the minimum cloudlet. Both the minimum and maximum cloudlets are considered fireflies. Eq. (2) calculates the attraction between two cloudlets by comparing any two cloudlets within the array. In Eq. (2), the parameter *r* represents the distance between the two cloudlets. This variable represents the largest waiting task, its waiting time, the queue of waiting tasks, and VM status in the single cloudlet. This distance *r* between the maximum and minimum cloudlets is calculated using Eq. (3). Steps 17 to 19 in the algorithm compare any two cloudlets. The corresponding columns of the two cloudlets in Table 2 are compared sequentially to evaluate their respective task and resource parameters. Each column of the first cloudlet is compared with the same column of the second cloudlet. Finally, the results of all comparisons of the columns are summed up for each cloudlet. Then, this sum is multiplied by the attraction found by Eq. (2), which gives the solution found in steps 17 to 22. This solution consists of the maximum and minimum cloudlets. After the maximum number of iterations, the algorithm gives the global maximum and global minimum cloudlets, which is found in steps 25 to 29.

Eq. (2) was modified from the standard Firefly algorithm by excluding the previous-iteration distance and cloudlet movement, as the cloudlets remain stationary. The distance *r* and previous results are not considered in the current iteration. Instead, the maximum and minimum cloudlets are sequentially identified from the stored array. The new solution to find the attraction between

the two cloudlets is given in Eq. (5), which is a modification of Eq. (2) of the Firefly optimization algorithm.

$$x_i^{t+1} = \beta_0 e^{-\gamma r_{ij}}(x_j^t - x_i^t) + \alpha\xi_i^t \quad (5)$$

In Eq. (5), the first part represents the attraction between the cloudlets, while the second part represents the control of random values when a global value is reached. The value for α is set to 1 and the lower and upper bound values determine the scale value using Eq. (6).

$$\xi = ub - lb \quad (6)$$

The scale value ξ represents the total number of fireflies found, with the lowest ranking firefly *lb* and the topmost firefly *ub*. In TALBMCC, ξ represents the total number of cloudlets found from the total number of rows in the table load_bal (*n*,p). Migration occurs from the cloudlet with the largest waiting task and the longest waiting time to the cloudlet with the shortest waiting queue. This process is repeated every second for load balancing. The technique reduces the cloudlet dropout rate, the response time, the task execution time, and the mobile power consumption, while improving the task execution by utilizing idle cloudlet resources. Task execution time and mobile power consumption are calculated using the formulas from the previous METATO study [14].

4 Performance Evaluation

The experimental set-up to evaluate TALBMCC consisted of development workstations, cloud servers, and distributed cloudlet service partners. The program was developed in C++ 20.0 using Visual Studio and executed through a dedicated UI with CGI. An i7, a 2.4-GHz laptop with 8 GB RAM was used as the workstation, while an i7 4-GHz cloud server with 16 GB RAM was leased through IaaS from i2k2.com to evaluate the existing and proposed methods. Task and mobile-device details were provided through the program. The proposed method, TALBMCC, was compared with other research techniques, namely, METATO, AGWO [15], HMCCALB [11], and MITBMCCE [10]. The comparison was done in three categories. In the first category, different numbers of tasks were used. In the second category, tasks from different velocities of mobile devices were used. In the third category, tasks from different numbers of mobile devices were used. These cases are explained in the following section.

4.1 Evaluation with Different Numbers of Tasks

The dropout rate of the cloudlet when using different numbers of tasks is shown in Figure 2. When comparing 1,000 tasks, the dropout rate for METATO was

30% while for TALBMCC it was 14%, which shows a 16% difference in the reduction of the dropout rate between the cloudlets.

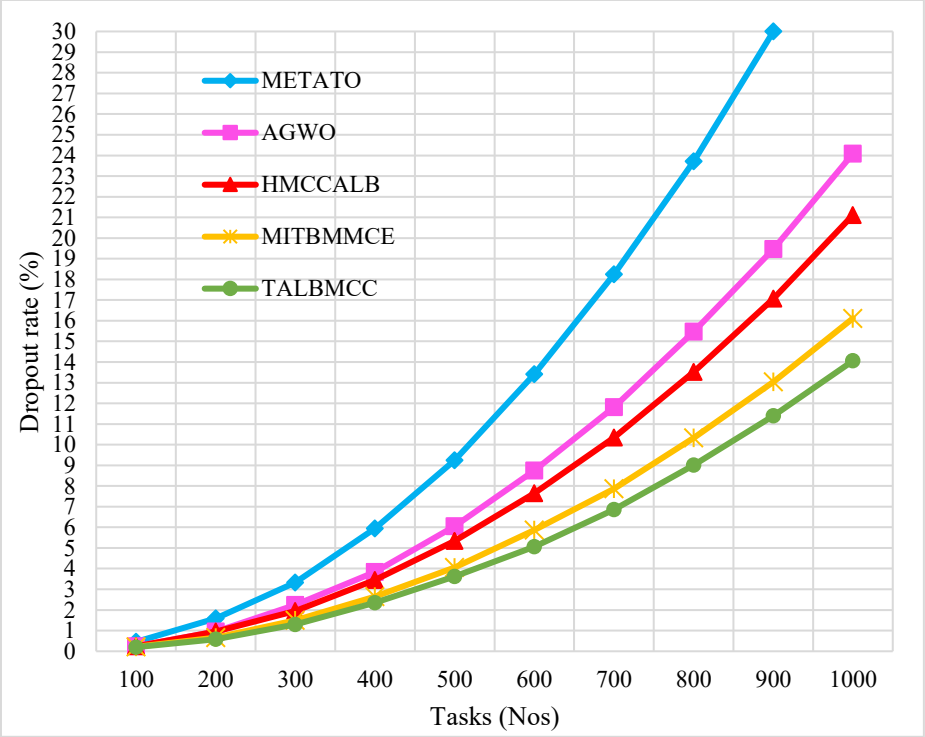

Figure 2 Dropout rate in % for different numbers of tasks.

The average response time of the cloudlets increased as the number of tasks increased, as shown in Figure 3. However, TALBMCC had a shorter response time when compared to the other strategies. Figure 4 demonstrates the number of cloudlets used by the task to complete its execution when it was offloaded from the mobile device. When the number of tasks increased, the offloaded tasks used more cloudlets. TALBMCC used idle VMs in less busy cloudlets better than busy ones in a single cloudlet. If there are more tasks, then also more cloudlets are used.

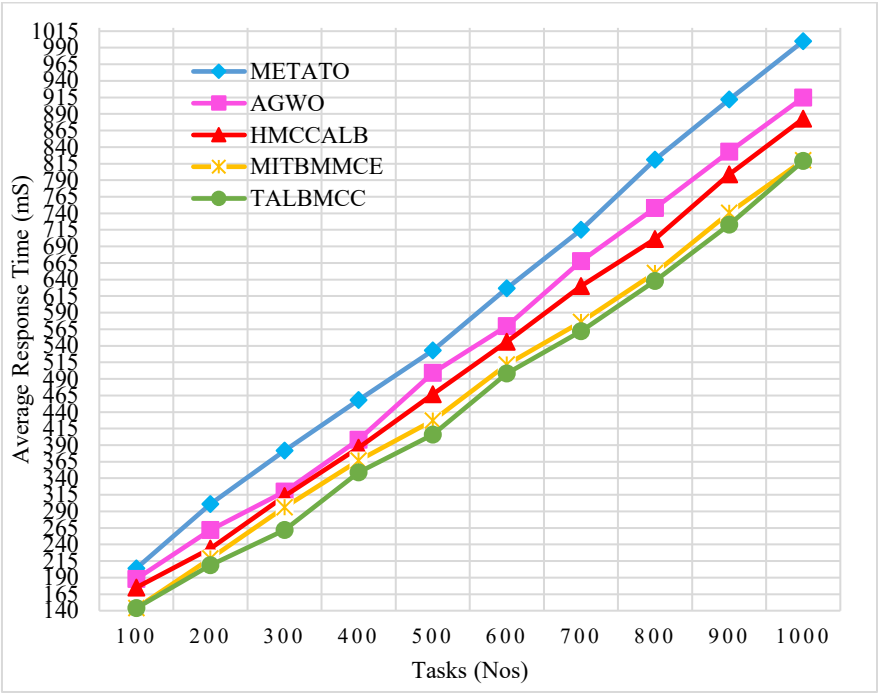

Figure 3 Average response time.

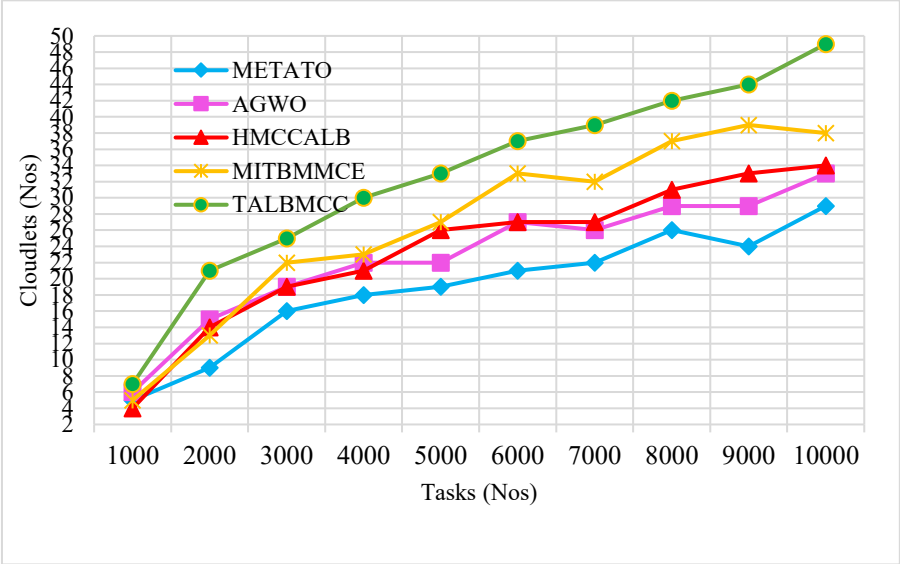

Figure 4 Number of cloudlets used by the task.

4.2 Evaluation with Different Velocities of Mobile Devices

Figure 5 shows the number of cloudlets used by the tasks offloaded for the different velocities of the mobile devices. The number of cloudlets increased with increasing speed. METATO only offloads and migrated a task when it is necessary. AGWO schedules the task within a single cloudlet and migrates the task when it is necessary. TALBMCC switches the task from one cloudlet to another to utilize the resources of every other cloudlet, causing an increase in the number of cloudlets used for completing the execution of the offloaded task.

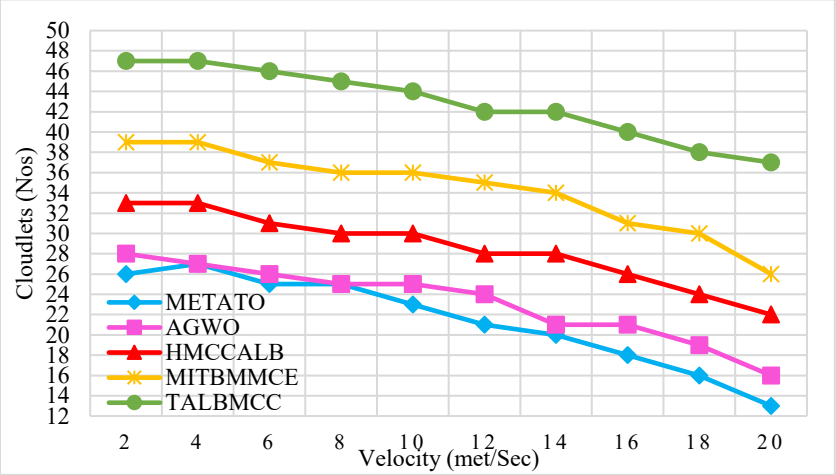

Figure 5 Number of cloudlets used by offloaded tasks from mobile devices with different velocities.

Figure 6 depicts the scheduling delay of the cloudlets when tasks were offloaded from mobile devices with different traveling speeds. TALBMCC had a shorter delay in scheduling the tasks.

Figure 7 shows the average execution time taken by the tasks offloaded for different speeds of the devices. The average execution time was more reduced with TALBMCC than with the other methods. The task's waiting time decreased since it was assigned to the cloudlet's idle VM, resulting in less execution time. Figure 8 compares the requests or tasks executed by the cloudlet when tasks are offloaded for different velocities of the mobile devices. TALBMCC prevents the largest task from holding resources for an extended period.

Figure 9 compares the average power consumption of the mobile devices when tasks were offloaded from different velocities of the mobile devices. The power consumption was reduced by TALMMCC because the results came quickly and the user could not switch to other mobile applications.

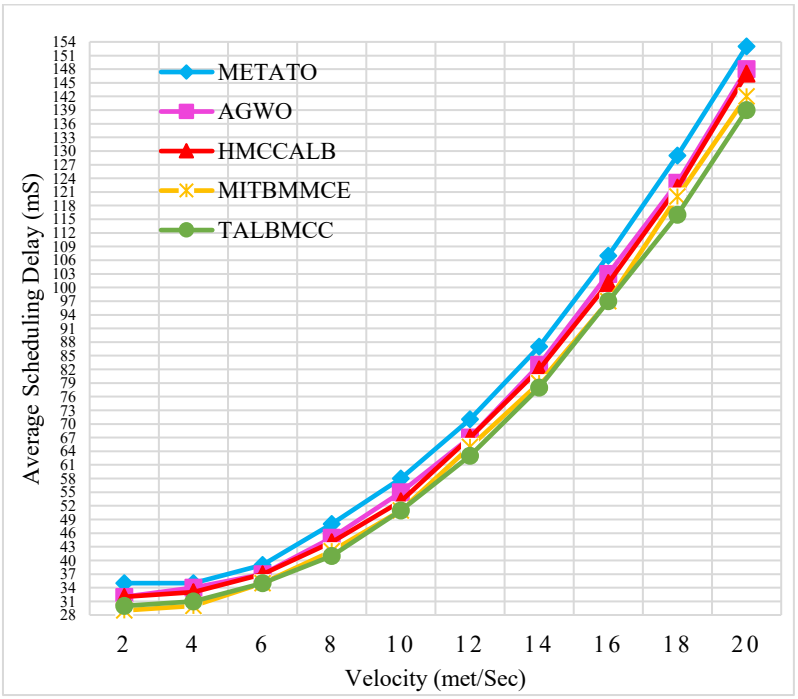

Figure 6 Average scheduling delay for offloaded tasks from mobile devices with different velocities.

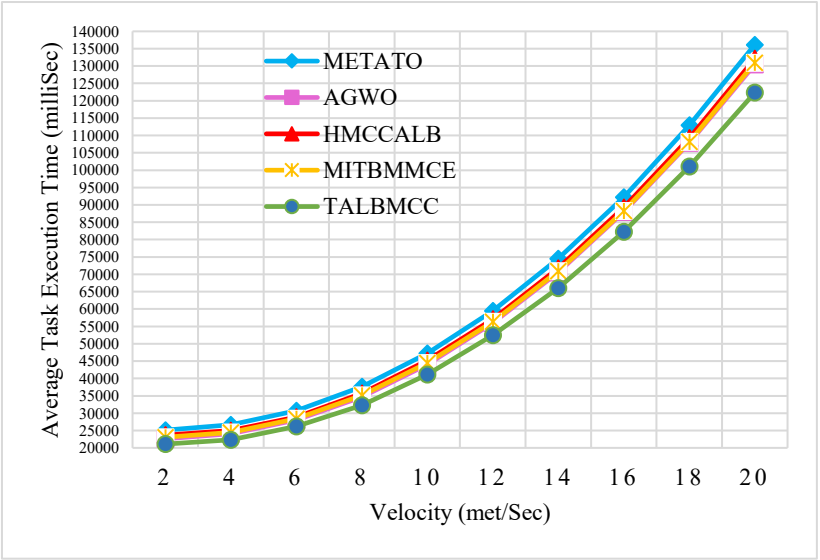

Figure 7 Average task execution time of offloaded tasks from mobile devices with different velocities.

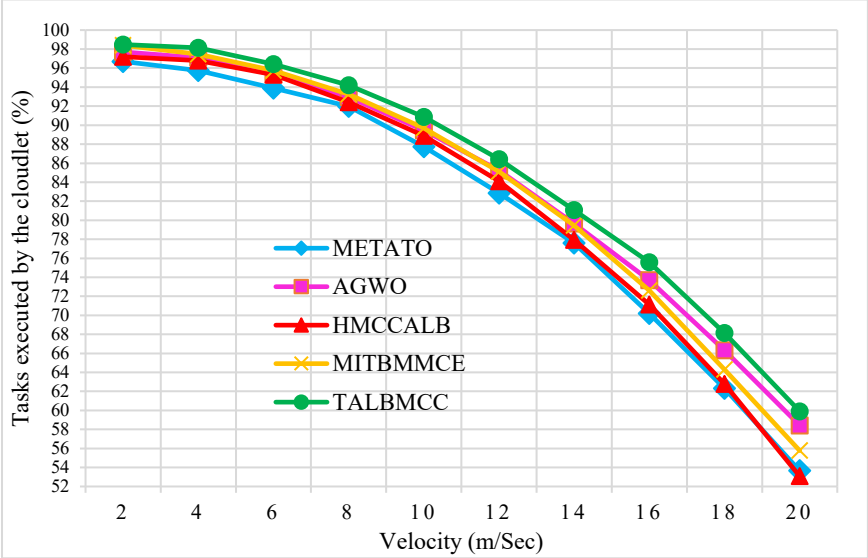

Figure 8 Tasks executed by the cloudlet from mobile devices with different velocities.

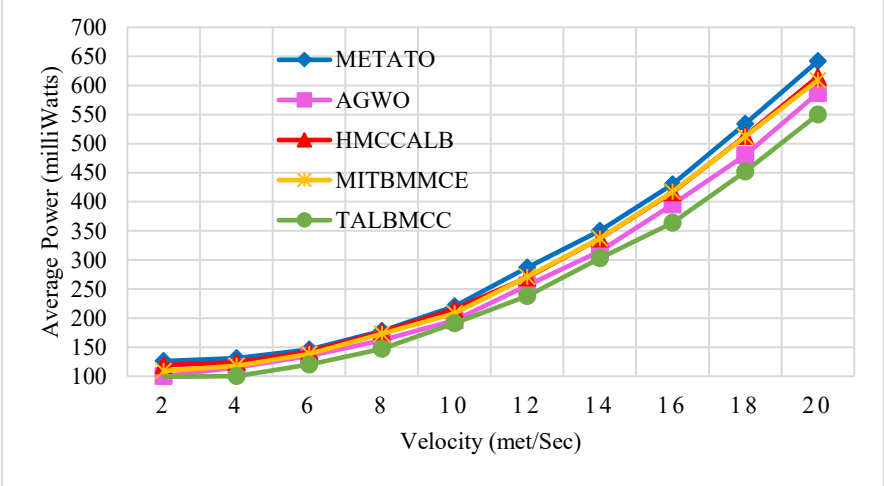

Figure 9 Average power consumption of mobile devices with different velocities.

4.3 Evaluation with Different Node Counts

Figure 10 shows the changing number of mobile devices that offloaded tasks to cloudlets, comparing TALBMCC with the other methods. TALBMCC lowered

the cloudlets' average scheduling delay. Migration of tasks to different cloudlets shortened the average scheduling delay of the cloudlets for allocation to the VM.

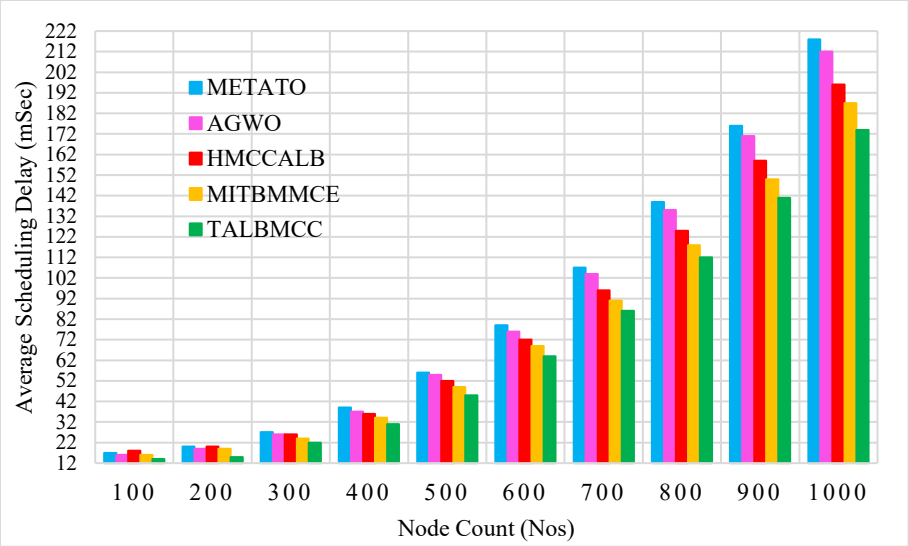

Figure 10 Average scheduling delay for offloaded tasks for different mobile node counts.

The average execution time taken by tasks offloaded from different counts of mobile devices is shown in Figure 11. TALBMCC reduced task execution times on average more than any other technique. Shorter transmission time reduces the average execution time of the task.

Figure 12 shows the tasks executed by the cloudlets when the tasks were offloaded from various counts of mobile devices. When the number of mobile devices surpassed 1,000, TALBMCC worked better than any other method.

Figure 13 shows the average power consumption for various numbers of mobile devices. TALBMCC reduced the average power consumption of the mobile devices when the node count exceeded 700.

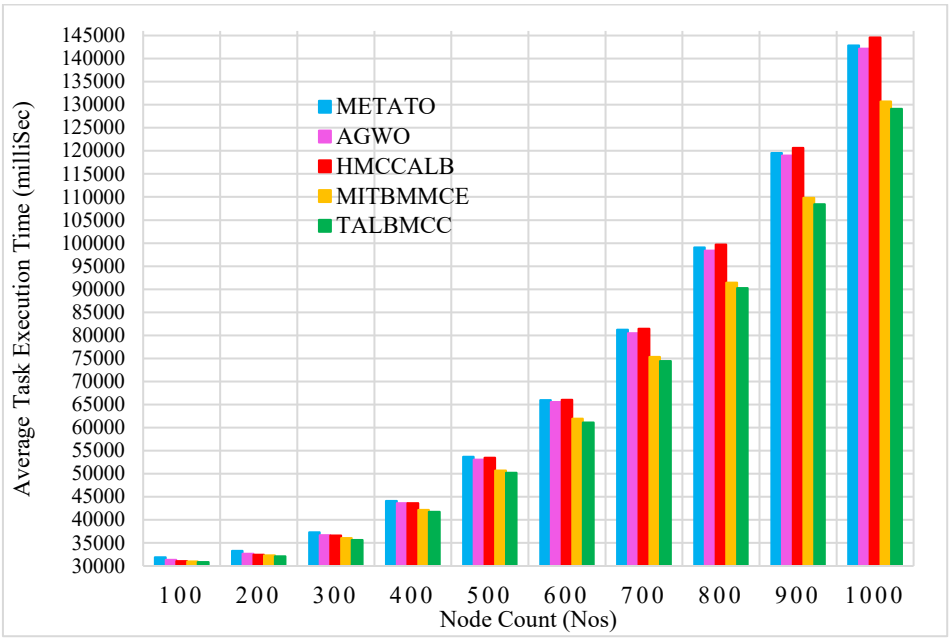

Figure 11 Average task execution time of offloaded tasks for different mobile node counts.

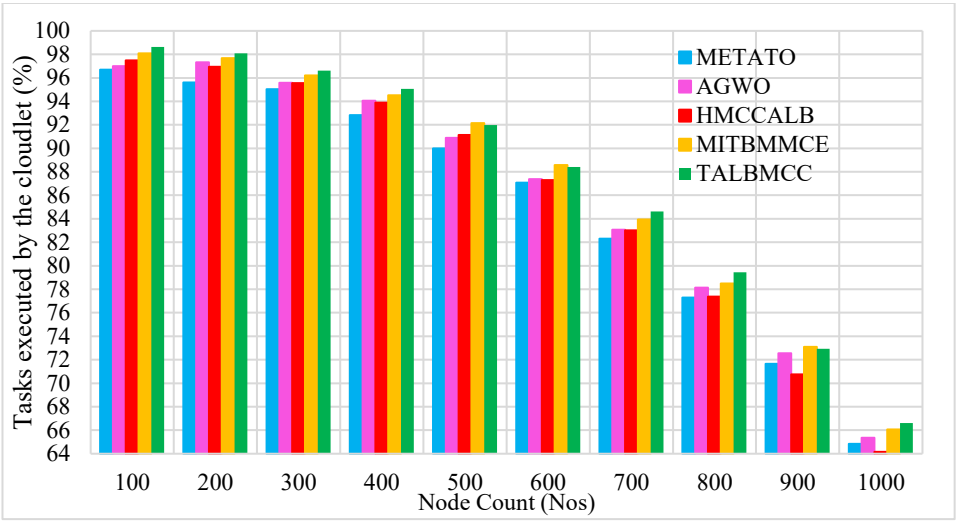

Figure 12 Tasks executed by cloudlets for different mobile node counts.

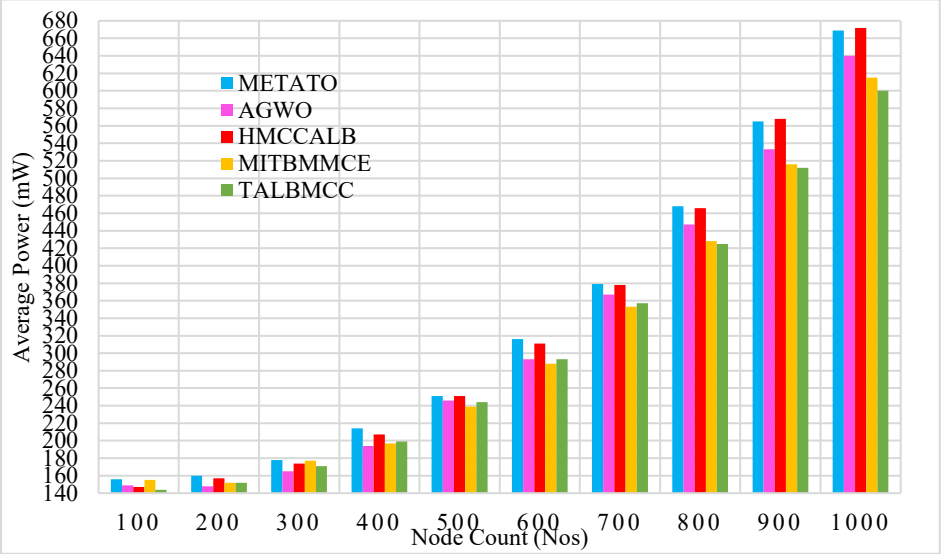

Figure 13 Average power consumption of different mobile node counts.

5 Conclusion

Task-Aware Load Balancing in Mobile Cloud Computing using Cloudlets (TALBMCC) outperformed any other technique in reducing the dropout rate and the response time of the cloudlets. It also successfully reduced the average scheduling delay of the cloudlets, the execution time of the offloaded task, the power consumption of the mobile devices and increased the tasks executed by the cloudlets. These parameters were achieved with any number of mobile devices and any speed of the mobile device. This method utilizes more cloudlets than other methods and migrates only the big tasks with the longest waiting time, thus causing small tasks to get a VM quickly. All the parameters used can be obtained from the different groups of data about the task and the cloudlets. To find the maximum cloudlet and the minimum cloudlet at the same time from these groups of data, TALBMCC uses the Firefly algorithm. To further improve the method and provide a better QoE to the user, it is necessary to concentrate on the development of parallel tasks in future work.

Conflicts of Interest

On behalf of all authors, the corresponding author states that there are no conflicts of interest to report.

Data Availability Statements

Since the contribution work was based on a live and dynamic cloudlet environment, the data was fetched directly from the leased cloud server.

References

[1] Satyanarayanan, M., Bahl, P., Caceres, R. & Davies, N., *The Case for Vm-based Cloudlets in Mobile Computing*, IEEE Pervasive Computing, **8**(4), pp. 14-23, October 2009.

[2] Cloudlets, *Cloudlet-specific Match Examples*, Available: https://techdocs.akamai.com/cloudlets/v2/reference/cloudlet-match (24-May-2023).

[3] Sinky, H., Khalfi, B., Hamdaoui, B. & Rayes, A., *Responsive Content-centric Delivery in Large Urban Communication Networks: A LinkNYC Use-Case*, IEEE Transactions Wireless Communications, **17**(3), pp. 1688-1699, 2018.

[4] Yang, X.S., Firefly Algorithm (chapter 8). *Nature-inspired Metaheuristic Algorithms*, Luniver Press, Cambridge, **172**, pp. 79-90, 2008.

[5] Lai, S., Fan, X., Ye, Q., Tan, Z., Zhang, Y., He, X. & Nanda, P., *FairEdge: A Fairness-Oriented Task Offloading Scheme for IoT Applications in Mobile Cloudlet Networks*, IEEE Access, **8**, pp. 13516-13526, 2020.

[6] Herbert Raj, P., Ravi Kumar, P. & Jelciana, P., *Load Balancing in Mobile Cloud Computing using Bin Packing's First Fit Decreasing Method*, Springer Nature, **888**, pp. 97-106, 2019.

[7] JongBeom Lim & DaeWon Lee, *A Load Balancing Algorithm for Mobile Devices in Edge Cloud Computing Environments,* Electronics, **9**(4), 686, 2020. doi:10.3390/electronics9040686

[8] Ranapana, R.A.A.I.B. & Jayasena, K.P.N., *Novel Approach for Load Balancing in Mobile Cloud Computing*, 6th International Conference on Information Technology Research (ICITR), Moratuwa, Sri Lanka, 2021, pp. 1-6, doi: 10.1109/ICITR54349.2021.9657441, 2021.

[9] Arun, C. & Prabhu, K., *An Efficient Job Sharing Strategy for Prioritized Tasks in Mobile Cloud Computing Environment using ACS-JS Algorithm*, Journal of Theoretical and Applied Information Technology, **97**(4), 2019.

[10] Hussein, K.Q., *A Multimedia Information Time Balance Management in Mobile Cloud Environment Supported by Case Study*, International Journal of Interactive Mobile Technologies, **16**(19), pp.118-132, 2022. doi: 10.3991/ijim.v16i19.33615

[11] Lee, A., Mhatre, J., Das, R.K. & Hong, M., *Hybrid Mobile Cloud Computing Architecture with Load Balancing for Healthcare Systems*, Computers, Materials & Continua, **74**(1), pp.435-452, 2023. doi: 10.32604/cmc.2023.029340

[12] Kanbar, A.B. & Faraj, K., *Region-Aware Dynamic Task Scheduling & Resource Virtualization for Load Balancing in IoT–fog Multi-cloud Environment*, Future Generation Computer Systems, **137**, pp. 70-86, 2022. doi: 10.1016/j.future.2022.06.005

[13] Fister, I., Yang, X.S., Fister, D. & Fister Jr, I., *Firefly Algorithm: A Brief Review of the Expanding Literature*, X.-S. Yang (ed.*), Cuckoo Search and Firefly Algorithm*, Studies in Computational Intelligence, **516**, pp. 347-360, 2014. doi: 10.1007/978-3-319-02141-6_17

[14] Mary, J.A. & Aloysius, A., *Mobility and Execution Time Aware Task Offloading in Mobile Cloud Computing*, International Journal of Interactive Mobile Technologies (iJIM), **16**(15), pp. 30-45, Aug, 2022. doi:10.3991/ijim.v16i15.31589.

[15] Mary, J.A. & Aloysius, A., *Task Scheduling with Altered Grey Wolf Optimization (AGWO) in Mobile Cloud Computing using Cloudlet*, Journal of Communications Software and Systems, **19**(1), pp. 81-90, Jan. 2023. doi: 10.24138/jcomss-2022-0151.