Wireless Personal Communications (2023) 129:323–344
https://doi.org/10.1007/s11277-022-10099-0

HFTO: Hybrid Firebug Tunicate Optimizer for Fault Tolerance and Dynamic Task Scheduling in Cloud Computing

Manikandan Nanjappan[1] · Gobalakrishnan Natesan[2] · Pradeep Krishnadoss[3]

Accepted: 11 October 2022 / Published online: 2 December 2022

Abstract
Task scheduling is an important issue in cloud computing when it comes to achieving multiple goals and satisfying different user needs. The increasing demand and users urge the necessity to minimize the task completion time and enhance the load balancing capacity. To achieve this goal, this article proposes a Hybrid firebug and Tunicate Optimization (HFTO) algorithm. Based on the previous scheduling information, the HFTO classifier classifies the task and creates different variants of Virtual Machine (VM). This step helps to minimize the time taken for VM creation. The proposed HFTO task scheduling framework aims at optimizing different Quality of Service (QoS) parameters such as fault tolerance, response time, efficiency, and makespan. The optimization algorithm helps to expand the search space of the solutions and frames an optimal task scheduling strategy for the virtual machines. The HFTO optimization method has several advantages, including enhanced search capability and faster convergence. The HFTO algorithm improves the fault tolerance capability by allocating the tasks to appropriate resources based on the resource load peak. The lightweight tasks can be allocated to the resources with high CPU utilization and the computation-intensive tasks can be allocated to the resources with low CPU utilization. The response time and execution time are improved by task pre-emption. Hence the time complexity and computational complexity can be improved by the HFTO algorithm even with limited resource capability. The experiments are conducted using the CloudSim experimental platform and the results are compared to the state-of-art techniques. The performance of the proposed methodology is evaluated in terms of different performance metrics namely makespan, load balancing, and average execution time. The results show that, when compared to existing techniques, the proposed methodology provides higher load balancing efficiency and improved cloud task scheduling performance.

Keywords Cloud computing · Dynamic scheduling · Bio-inspired algorithm · Optimal scheduling · Load balancing · Firebug optimization

✉ Manikandan Nanjappan
macs2005ciet@gmail.com

Extended author information available on the last page of the article

1 Introduction

Users now have immediate access to pooled resources thanks to a new technology called cloud computing. The core of a cloud computing (CC) data center has changed as a result of the rapid growth of cloud computing technology. Cloud computing and supercomputing are made feasible by increasingly dispersed computing resources, such as high-bandwidth networks, massive data centers, and volumes of storage that are immense [1]. The daily rise in the number of cloud-based services that meet customer demands is made possible by the performance of the cloud environment. Some of the stages that the CC goes through include grid computing, utility computing, parallel computing, and distributed computing [2].

Among the many issues in cloud computing, the fundamental process at the Software as a service (SaaS) level is task scheduling [3]. The service provider must use a type of scheduling that maximizes resource usage while satisfying the demands of the clients. When tasks are completed simultaneously, the service provider distributes multiple tasks to a single or several VMs. While the cloud environment experiences an overall variance in demand, the static load balancing strategy is crucial [4]. Static algorithms cannot function in the cloud environment because workload changes over time. Hence, the workload balancing among the VMs needs dynamic methods. The scheduling process and NP-hard problem consider task scheduling based on the dynamic and heterogeneity properties of CC [31–33].

A new branch of networked computing called cloud computing disseminates data technical services across the Internet. More capable computing offers centralized storage, memory, computation, and bandwidth [5]. The backbone of CC is virtualization which enables virtual machines (VMs). In the cloud computing environment, the VMs execute the tasks presented by the clients. Because of the nature of the dynamic cloud, there is variation in physical machine (PM) load [6, 30]. The workload across many computing resources including disk drives, CPUs, network links, and computers is distributed via load balancing.

Dynamic and static are the two basic kinds of load balancing algorithms [28, 29]. At the runtime, the distribution of workload is changed by making the dynamic load balancing model. The dynamic load balancing methodology is used due to the unpredictability of workload[7]. The increasing demand and users urge the necessity to minimize the task completion time and enhance the load balancing capacity. The NP-complete problem considers the heterogeneous environment-based task scheduling [19–23, 34, 35]. A faster response time enhances system performance overall, and a faster completion time is crucial. According to the rapid development of data centers, energy efficiency is a prominent contradiction in the cloud. Nevertheless, it may lead to a few problems such as performance reduction and service response delay [24, 25]. It is critical to achieving a better balance between energy and performance. The major steps involved in this study are delineated as follows:

- A hybrid firebug and Tunicate Optimization (HFTO) algorithm is proposed for task scheduling and load balancing in the cloud.
- The HFTO algorithm is a combination of the Firebug Swarm Optimization (FSO) algorithm and Tunicate Swarm Optimization (TWO) algorithm in which the HFTO algorithm method has several advantages, including enhanced search capability and faster convergence.

- The HFTO algorithm minimizes the time taken for VM creation and optimizes different Quality of Service (QoS) parameters such as fault tolerance, response time, efficiency, and makespan.
- The proposed HFTO algorithm improves the time complexity and computational complexity.

The remainder of this article is organized as follows: Sect. 2 provides a review of the current works. The Hybrid Firebug and Tunicate Optimization (HFTO) algorithm are formulated in Sects. 3, and 4 presents the challenges that are the focus of this study. Section 5 gives an overview of the proposed work, while Sect. 6 discusses the experimental findings. Section 7 concludes the article.

2 Related Works

Ebadifard et al. [8] developed an autonomous task scheduling approach for load balancing in cloud computing. An increasing amount of inter-VM communication overheads is the major challenge in dynamic load balancing. This method's performance is evaluated by applying the CloudSim tool and then compared with Naive Bayesian Load Balancing, Honey-Bee, Autonomous, and Round Robin techniques. The experimental results demonstrated maximum load balancing and minimal communication overheads, as well as shorter response times and higher resource usage. However, geographical clouds with dispersed Datacenters are not taken into account.

Hybrid Tabu–Harmony task scheduling (HTHTS) algorithm was proposed by Alazzam et al. [9]. The Harmony and Tabu search algorithms were merged to improve the quality of the findings. While compared to the round-robin, Harmony search, and Tabu search methods, this task scheduling method yielded better results in the case of the total cost, makespan, and throughput. While throughput and cost are increased, task scheduling performance is hindered by crowded VMs.

The Service level agreement-based Load Balancing (SLA-LB) algorithm was suggested by Lavanya et al. [10]. The overall makespan of the task is reduced to exaggerate the performance of task scheduling in clouds. The expected time to complete (ETC) generates the threshold data. The expected robust threshold value optimizes load balancing across machines with the minimum configuration machine allocation to the task, as well as system utilization. Compared to the existing algorithms, the SLA-LB algorithm provided better results using the measures such as VM utilization, gain cost, penalty, and makespan. The experimental results showed that improved task allocation was associated with poor resource utilization and complicated costs. The new energy-efficient load balancing global optimization model was introduced by Lu et al. [11]. The load balancing and energy consumption optimization is their major objective function. Based on the experimental investigation, the minimum energy consumption with better power management is obtained but this model is not suitable for large data centers.

Khorsand et al. [12] introduced an energy-efficient task-scheduling (EETS) algorithm in cloud computing. Based on the best–worst Method (BWM), an energy-efficient task-scheduling model is introduced to determine cloud scheduling solutions. For each criterion, the importance weights are assigned by applying the BWM process. The effectiveness of this strategy is evaluated using a variety of statistical testing standards, including ANOVA and the CloudSim toolkit. Nevertheless, it does not consider reliability and computational

complexities. An Improved Firework Algorithm was proposed by Wang et al.[13] for task scheduling in cloud computing. The primary choice is lightweight computing because of its weak processing ability and fewer computing resources. The experimental results demonstrated better load balancing with minimum task processing time and it does not share the resources among data centers. Table 1 lists the summary of related works.

3 Formulation of Hybrid Firebug and Tunicate Optimization (HFTO) Algorithm

The Firebug Swarm Optimization (FSO) and Tunicate Swarm Optimization (TWO) algorithms are combined to form the Hybrid Firebug and Tunicate Optimization (HFTO) algorithm. As a result, the following is a breakdown of the HFTO algorithm's steps:

3.1 Firebug Swarm Optimization (FSO) Algorithm

The two basic behaviors of the Firebug Swarm Optimization (FSO) algorithm are gregarious behaviors and solitary individuals forming aggregations. These aggregations help select compatible partners for reproduction and lessen predator activity [14]. The Firebug behaviors of FSO algorithm models are explained below:

- Male firebugs use pheromones, which are chemical messages, to defend and attract the female bug colonies.
- The fittest female in its colony only mates with every male bug.
- The male bug based on the colony strongly attracts the female bugs.
- Fit females' chemical signals attract male bugs who would never join their colony.
- Even if a single aggregation moves together, the swarm members never disperse.

The following section explains the mathematical model of the FSO algorithm.

3.1.1 Female Colonies Formation

The fit bugs are connected with the lower cost values because the firebugs search for fit mates. In the search space, the FSO model initializes with the female bugs $M_N M_G$ in which every male bug contains the female bugs colony M_G. The uniform random vector variable treats each female bug's initial location in accordance with the search space.

3.1.2 Selection of Males

The placement of each male insect is first determined by the best female bug in the colony, but the male bug mates with the healthiest female bug. Every male bug position is updated by the best female in the colony. The location updates are implemented using operations on elements like Hadamard matrix multiplication.

3.1.3 Female Bug Chemotactic Movement

According to the male selection behavior, the female bug's location is updated after initialization. The FSO strategy relies heavily on avoiding scalar operations in order to achieve

Table 1 List of related works

References	Methods	Parameters	Advantages	Disadvantages
Ebadifard et al. [8]	The autonomic task scheduling algorithm	Makespan and communication overhead	Maximum load balancing degree with minimum communication overheads	Geographic clouds with distributed data centers were overlooked
Alazzam et al. [9]	Hybrid Tabu–Harmony task scheduling algorithm	Cost, makespan, and throughput	Better throughput and lower cost	Overloaded VMs reduces the task scheduling performance
Lavanya et al. [10]	Service level agreement-based Load Balancing algorithm	VM utilization factor, gain cost, penalty, and makespan	Better task allocation	Poor resource utilization with complex cost
Lu et al. [11]	New energy-efficient load balancing global optimization model	Energy consumption, resource utilization, and makespan	Minimum energy consumption with better power management	Not suitable for large data centers
Khorsand et al. [12]	Multi-criteria decision-making method	Resource utilization, energy consumption, and makespan	Independent tasks	Does not consider reliability and computational complexities
Wang et al. [13]	Improved Firework Algorithm	Processing time and throughput	Better load balancing with minimum task processing time	Not sharing resources among datacenters

a successful result. Store all of the female bug positions in one matrix corresponding to a specific male colony. Let $n(n) \cdot G$ be the corresponding female bug position stored in a matrix M_G. The effective Hadamard multiplication model with a matrix update equation updates the female bugs.

$$N_y \leftarrow repmat(n(n) \cdot y, 1, M_G) \tag{1}$$

$$N_z \leftarrow repmat(n(b) \cdot y, 1, M_G) \tag{2}$$

The random integer b falls in interval 1 and M_G. With the function *repmat* (B, m, n), the n copies of B return a matrix in addition to the row dimension.

$$n(n) \cdot G \leftarrow n(n) \cdot G + C_1\Theta\left(N_y - n(n) \cdot G\right) + C_2\Theta\left(N_z - n(n\_ \cdot G\right) \tag{3}$$

3.1.4 Male bug's Attraction Toward Fittest Female Bugs

The fit females attract every male bug. The male bug moves to a comparable fittest bug based on the specific geographical position, and the entire population is limited to the specified geographical position, preventing the swarm from dispersing. Equation (4) explains the update rule for the male bug movement behavior to the fittest female bug.

$$n(n) \cdot G \leftarrow n(n) \cdot y + C_3\Theta(h - n(n)y) \tag{4}$$

3.1.5 Swarm Cohesion

Individual bugs are not scattered in a swarm when the entire swarm moves as a single entity [15]. The herd cohesion model is represented in Eq. (5).

$$n(n) \cdot y \leftarrow n(n) \cdot y + C_4\Theta(h - n(a)y) \tag{5}$$

Individual bugs must choose the best match for reproduction based on the reproduction swarm behavior of firebugs. Based on the triangular vector law, the y^1 and y^2 express the vector y is expressed as:

$$y = y^1 + \beta(y^2 - y^1) \tag{6}$$

The relocation of the male insect to the fittest female bug is accomplished by Eq. (7).

$$n(n) \cdot y \leftarrow n(n) \cdot y + \beta(h - n(n) \cdot y \tag{7}$$

The element-wise Hadamard multiplication that takes the place of the scalar vector is shown in Eq. (8).

$$n(n) \cdot y \leftarrow n(n) \cdot y + C_4\Theta(h - n(a) \cdot y \tag{8}$$

The below expression delineates the weak movement of female bugs to random males and the strong movement to the dominant.

$$n(n) \cdot G \leftarrow n(n) \cdot G + C_1\Theta(N_y - n(n) \cdot G + C_2\Theta\left(N_z - n(n) \cdot G\right) \tag{9}$$

The movement to the random male and the dominant male is denoted as $C_2\Theta\left(N_z - n(n) \cdot G\right)$ and $C_1\Theta(N_y - n(n) \cdot G$. Additionally, the degree of attraction to dominant and random male bugs is C_1 and C_2.

3.2 Tunicate Swarm Optimization (TSO) Algorithm

The conflict among the search agents avoidance is performed via a vector to calculate the position of the new search agent [15].

$$\vec{P} = \frac{\vec{H}}{\vec{M}} \tag{10}$$

$$\vec{H} = a_2 + a_3 - \vec{F} \tag{11}$$

$$\vec{F} = 2.a_1 \tag{12}$$

The gravity force and the water flow advection are $\vec{H}$ and $\vec{F}$. The random interval 0 and 1 for the random variables such as a_1,a_2 and a_3. Where $\vec{M}$ express the social force between the search agents.

$$\overrightarrow{M} = [S_{\min} + a_1.S_{\max} - S_{\min}] \tag{13}$$

From Eq. (13), $S_{\max}$ and $S_{\min}$ are the subordinate and initial speeds. The values of $S_{\min}$ and $S_{\max}$ are 1 and 4.

3.2.1 Follows the Optimal Agent Position

For reaching the optimal solution, following the current best agent is important. There is no fighting among the agents in the swarm. Equation (14) describes the best agent with its optimal position [16].

$$\overrightarrow{RD} = \left| Y_{best} - r_{andom} \cdot \overrightarrow{S_p(y)} \right| \tag{14}$$

From the above equation, the food source and search agent distance is $\overrightarrow{RD}$. Where, Y_{best} and y is the food source position and the current iteration of tunicate. The 0 to 1 interval range for the random integer (r_{andom}) and $\overrightarrow{S_p(y)}$ is the position of the tunicate.

3.2.2 Keep Near to the Best Agent

Equation (15) computes the best position to ensure the search agent is still near to the best agent. The best food source position Y_{best} is and the tunicate position is $\overrightarrow{S_p(y)}$.

$$\overrightarrow{S_p(y)} = \begin{cases} Y_{best} + \vec{P} \cdot \overrightarrow{RD} \;\; if \;\; r_{andom} \geq 0.5 \\ Y_{best} - \vec{P} \cdot \overrightarrow{RD} \;\; if \;\; r_{andom} < 0.5 \end{cases} \tag{15}$$

3.2.3 Swarm Behavior

Based on the positions of two agents, update the current agent position to model the swarm characteristics of tunicates.

$$\overrightarrow{S_p(y+1)} = \frac{\overrightarrow{S_p(y)} + \overrightarrow{S_p(y+1)}}{2 + a_1} \tag{16}$$

Algorithm 1: Pseudocode of HFTO algorithm.

Initialize the HFTO algorithm parameters with the maximum number of iteration

For $\leftarrow$, M_N **do**

$\cdot G_f$)

) (min max $y_{\min} \times$ *random*–*E*)M_G)

$ind, val] \leftarrow \min(\cdot G_f)$

) $y(\leftarrow val$

$n(n)$ y $n(n) \cdot G(:ind)$

End For

h m $ind) \cdot y$

$h_f \leftarrow val$

For $\leftarrow$, $S1_{\max}$ **do**

Formulate the female colonies

Select male firebugs

Chemotactic movement of female firebug

Use equation (4) to update the rule for the male bug movement behavior to the fittest female firebug

Equation (5) delineate the herd cohesion model

Update the behavior of the tunicate swarm equation (16)

Obtain the optimal solution

End For

3.3 HFTO Algorithm

The Firebug Swarm Optimization (FSO) algorithm has several advantages such as reducing execution time, modern CPUs, and GPUs of fast Single Instruction Multiple Data (SIMD) feature in which it performs multiple arithmetic operations, lower cost, exploration, and Behaviour and etc. However, the FSO algorithm's exploitation swarm cohesiveness behavior required few changes because it degrades FSO's performance in terms of convergence speed, searchability, and computational complexity. As a result, we integrated the swarm behavior of tunicate swarm optimization to the FSO exploitation swarm cohesion, so improving the FSO algorithm's performance, and the newly developed model is known as the Hybrid Firebug and Tunicate Optimization (HFTO) method. As a result, the HFTO approach offers various advantages, such as improved search capacity and faster convergence. Algorithm 1 delineates the pseudocode of the HFTO algorithm.

4 Problem Formulation

The set of N number of virtual machines is described as $VM = \{VM_1, VM_2, VM_3, ..., VM_N\}$. Where $TA = \{TA_1, TA_2, TA_3, ..., TA_K\}$ describes the set of K tasks. We need to know how much the load has changed amongst the virtual machines to do load balancing [17]. The single virtual machine of CPU capacity (CA_j) is expressed in Eq. (17).

$$CA_j = NIRP_j \tag{17}$$

Equation (18) describes the capacity (C) of the physical machine.

$$C = \sum_{j=1}^{m} CA_j \tag{18}$$

The below equation calculates the load on single virtual machines.

$$L_{VMj} = \sum_{k=1}^{i} SL_k \tag{19}$$

Equation (19) calculates the load on single physical machines.

$$L = \sum_{j=1}^{n} L_{VMj} \tag{20}$$

The below equations explains the processing time of virtual machine and physical machine.

$$RS_j = \frac{L_{VMj}}{CA_j} \tag{21}$$

$$RS = \frac{L}{C} \tag{22}$$

The host is in a balanced state based on the value of the RS parameter. Equation (23) computes the workload standard deviation.

$$\sigma = \sqrt{\frac{1}{n} \sum_{j=1}^{n} (RS_j - RS)} \tag{23}$$

The priority for each task is applied to any virtual machines based on the physical machines [18]. Compared with the computed standard deviation σ, the virtual machine is in a balanced state when the task is executed. the threshold is set from 0 to 1 and the standard deviation is within the threshold limit.

5 Proposed HFTO Algorithm for Resource Allocation and Load Balancing in the Cloud

The loads are balanced in the cloud data center and the optimum resource is assigned using the proposed HFTO algorithm. Virtual machines that are present on a single physical host are the subject of intrahost load balancing [19]. The mapping across load scenarios is balanced as a result of a thorough examination of the HFTO algorithm. Table 2 describes the relationship between the HFTO algorithm scheduling scenario and its behavior.

5.1 Virtual Machine Classification

The proposed HFTO algorithm with its working process is delineated in Fig. 1. On the basis of the proposed model, load balancing tasks are removed from the workload underloaded VM, overloaded VM, and balanced VM. The VMs are mainly grouped based on their upper and lower boundaries. The upper boundary comprises 70% of the physical machine capacity whereas the lower boundary comprises 20% of the physical machine capacity. The overloaded VMs are represented using the $\left(L_{VMj} > 0.8 * C\right)$ and the underloaded VM are expressed using the $\left(L_{VMj} < 0.3 * C\right)$ criteria. The tasks are balanced by taking the tasks waiting in the overutilized VM and scheduling them in the underutilized VM. The priority and residual completion time are two essential factors that must be taken into account in order to reduce the task's makespan and response time.

The proposed HFTO task scheduling paradigm seeks to maximize Quality of Service (QoS) metrics, including fault tolerance, response time, effectiveness, and makespan. The high-priority tasks are mainly taken both for pre-emption and exemption. The task is mainly exempted in a scenario where a high-priority task is executing in a particular VM to minimize the failure in terms of QoS on the service provider side. In this way, the Service Level Agreement (SLA) is not violated.

5.2 Task Scheduling

The optimal task scheduling is carried out using the HFTO algorithm. The main challenge with task scheduling is determining how to execute jobs that have been removed from virtual machines. The tasks withdrawn from the VM are mostly assigned to the new VM, which has a lower task load and tasks with varying priority. Equation (24) frames the objective function.

Table 2 The mapping among the HFTO algorithm scheduling scenario with its behavior

Behavior	Less loaded group of VM
Tandem formation	Capacity and loan on every virtual machine
Female firebug population	Tasks neglected from a virtual machine belong to the overloaded virtual machine group
Dominant male firebug	Tasks in new or queue incoming tasks
Male firebug	Running tasks in every virtual machine
Female firebug chemical signal	A virtual machine in an underloaded VM group

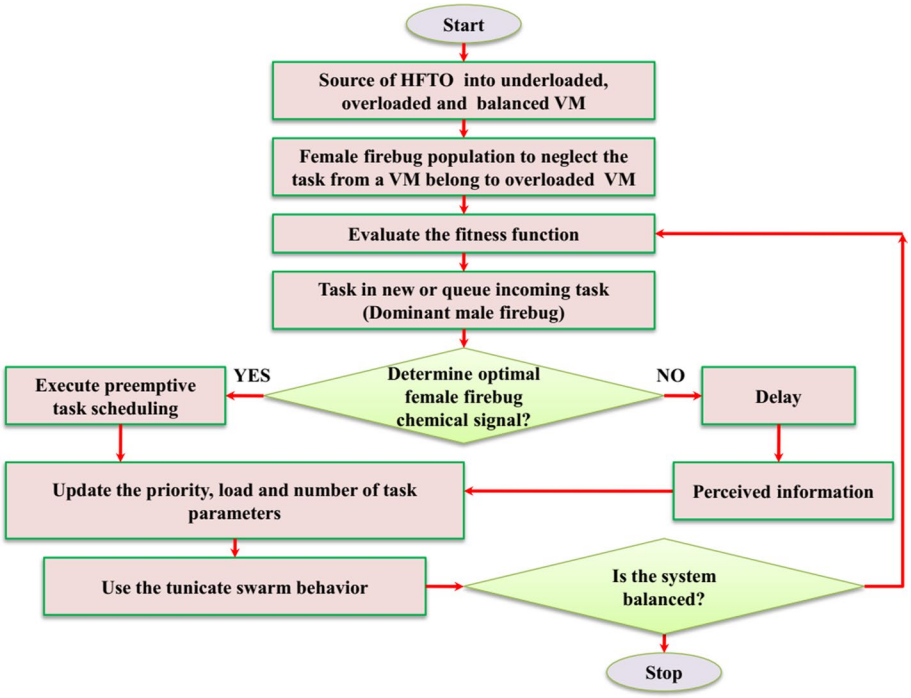

Fig. 1 Proposed HFTO algorithm for task scheduling and load balancing

$$F(y) = \min\left(MS_t + L_{VM_j}\right) \tag{24}$$

The load on the VM is L_{VM_j} and the total number of tasks in the queue is MS_t. The randomly dispersed solution in the search space is derived by Eq. (25).

$$y_{j,k} = y_{j,\min} + random[0,1] \cdot \left(y_{j,\max} - y_{j,\min}\right) \tag{25}$$

The upper and lower bounds with respect to $y_{j,k}$ are $y_{j,\max}$ and $y_{j,\min}$. Based on the overloaded virtual machine groups, the female firebug updates the position by applying Eq. (26) after generating the initial population.

$$u_{j,k} = y_{j,k} + \lambda_{j,k} \cdot \left(y_{j,k} - y_{i,k}\right) \tag{26}$$

Equation (27) delineate the fitness function of a male firebug after updating the position.

$$R_j = \frac{F(y_j)}{\sum_{j=1}^{m} F(y_j)} \tag{27}$$

The population initializations of firebugs (FB) are described in Table 3. The matrix values are mainly the VM IDs present in the underutilized VM. The overall number of tasks largely indicates the tasks that were removed from the overused VMs. The selection probability of HFTO is higher when the fitness value is higher. The optimal scheduling is obtained after reaching the maximum iteration. However, the priority of the task running is considered when

Table 3 Population initialization

Number of tasks	FB_1	FB_2	FB_3	FB_4	FB_5	FB_6	FB_7	FB_8	FB_9	FB_{10}
T_1	3	1	1	1	3	1	2	3	2	3
T_2	3	1	2	3	3	1	1	3	2	3
T_3	3	2	1	2	1	2	3	1	1	1
T_4	2	2	1	2	2	1	3	2	3	1
T_5	1	3	2	2	2	3	1	2	2	1
T_6	2	1	2	3	1	1	1	1	1	1
T_7	1	1	3	2	3	2	2	1	3	2
T_8	2	3	3	1	1	1	1	3	2	2
T_9	3	2	1	1	1	2	3	1	3	1
T_{10}	2	2	1	1	2	3	3	1	3	2

performing preemptive scheduling. The optimal scheduling plan mainly comprises the ID of the VM where the removed task is executed.

5.3 Optimal Scheduling with Preemptive Scheduling

Preemption happens when the removed task's priority is higher than the next task waiting in the queue. In certain cases, the running task's remaining estimated completion time is longer than the eliminated task burst time. Once the requirement is met, the removed task is carried out by the virtual machine. The preempted task state is preserved by the checkpoint mechanism. If the priority of the incoming task is greater than the priority of the task currently running in the VM, then it is executed first, or else it waits in the target VM queue. Next, the execution time of both tasks is checked, if the running task execution time is lower than the prioritized task, then it is executed. The higher-priority task is completed first if its running time exceeds that of the remaining task; otherwise, its current state is maintained. After the task allocation process is complete, the VM is assigned several parameters, such as the number of tasks provided, their priority, updated overloaded and underloaded VMs, etc. We mainly formulate the proposed model on the assumption that at least one physical machine may fail. The task scheduling module runs completely before the deadline. The makespan is computed by taking the Module Processing Time (MST) and Data Transmission Time (DTT). The MST is the variation between the ending time and task execution deadline. The DTT and MST values are normalized via different weight coefficients x and y. Since the MST and DTT both have equal importance, the coefficients x and y are allocated the same weights.

$$MP = x * DTT + y * MST$$

The checkpoint restore model is used here for fault tolerance.

6 Result and Discussion

A cloud simulation toolset, including CloudSim [26, 27], supports both behavior and system models of cloud system components, including resource provisioning rules, virtual machines, and data centers. CloudSim also shows how well the proposed model performs. At least four virtual machines are running on each of the 20 hosts in a single data center.

The task length is in the 400–1 billion instruction range. When the virtual machines are started and scheduled to run, 300 cloudlets are created. With one CPU core, each node has minimum and maximum processing capabilities of 500 MIPS and 20,000 MIPS, respectively. 50 individuals comprise the population, and there is 100 iterations total. The parameters settings of the proposed model are described in Table 4.

6.1 Makespan Performance

Makespan mainly computes the overall time required to schedule the input tasks by selecting an optimal VM in the cloud and it is mathematically computed as follows:

$$N_j = \max\left\{CS_{j,k}/j \in S,\ \ j = 1, 2, ..., m\ \ and\ \ k \in virtual\,machines,\ \ k = 1, 2, ..., n\right\} \tag{28}$$

In terms of makespan, Fig. 2 depicts the results achieved by comparing the proposed model to conventional techniques. Depending on the makespan in seconds, the number of tasks ranges from 100 to 500. The proposed method efficiency is validated using the significant performance measure called makespan. The makespan is calculated using Eq. (28). The time taken with the virtual machines to finish each task execution determines the makespan. The state-of-art comparison is carried out using the proposed method with existing Hybrid Tabu–Harmony task scheduling (HTHTS) [9], Service level agreement-based Load Balancing (SLA-LB) [10], Energy-efficient task-scheduling (EETS) [12], and Improved Firework Algorithm (IFA) [13]. Because of the constant number of virtual machines, the number of jobs increased by increasing the execution time. The proposed

Table 4 Proposed parameter settings

Parameter	Ranges
Common and HFTO algorithm parameters	
Scheduling and load balancing model	HFTO algorithm
Number of data centers	1
Number of virtual machines	2–64
OS of datacenter	Linux
Bandwidth	1000–3000 MB
Number of tasks	100–500
Population size	50
Maximum number of iterations	100
VM setup of data center	
Bandwidth	
Bandwidth	100 M/s
RAM	4096 MB
Dist input output	10 GB
CPU computing capacity	1860 MIPs and 2660 MIPs
Task setup of data center	
Output size (memory)	[20, 40] MB
File size	[200, 1000] MB
CPU length	[400, 1000] MIPs

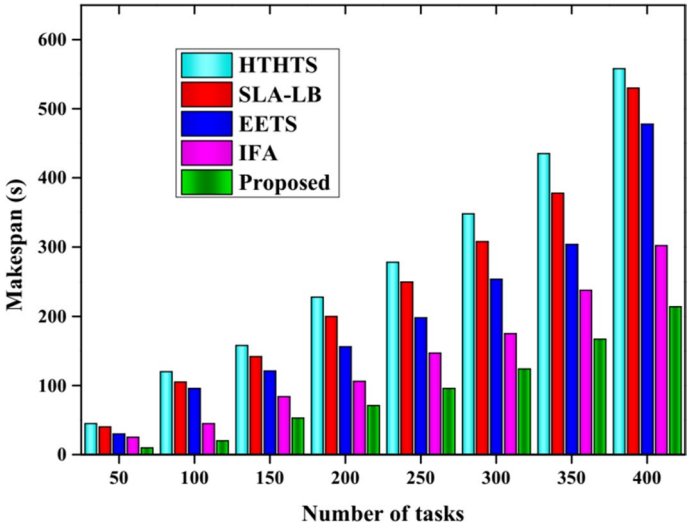

Fig. 2 State-of-art comparison of makespan

method demonstrated less makespan than other existing techniques such as HTHTS, SLA-LB, EETS, and IFA [28–30]. The proposed model minimizes the makespan by the heuristic-based load balancing algorithm created.

Figure 3 delineates the state-of-art result of makespan based on different task types. The four task types (T_j) are plotted on the horizontal axis such that $j \in \{1, 2, 3, 4\}$. When the task types changed from 1 to 4, so did the makespan values of the state-of-the-art approaches, including HTHTS, SLA-LB, EETS, and IFA with the proposed model. Compared to the existing methods, the proposed technique provided better makespan results based on varying makespan. The shorter the makespan, the better the service quality and the better the scheduling. From the below equation, the task completion time in the cloud is μ_j.

$$N = \frac{\min \sum_{j=1}^{M} \mu_j}{M} \tag{29}$$

6.2 Response Time Performance

A higher response time taken by a model specifies its weak capability of it in handling the overloaded VMs. Hence, a reliable model needs to have a minimal response time as possible to effectively manage the load in the network. Figure 4 displays a cutting-edge comparison of response time based on the number of tasks. Depending on the makespan in seconds, the number of tasks ranges from 100 to 500. For user satisfaction, the measurement criterion is response time. Equations (30) and (31) is used to calculate the response time. The low and medium-priority jobs are removed from overloaded virtual machines. When compared to the existing techniques, the amounts of task migrations are fewer. In comparison to existing

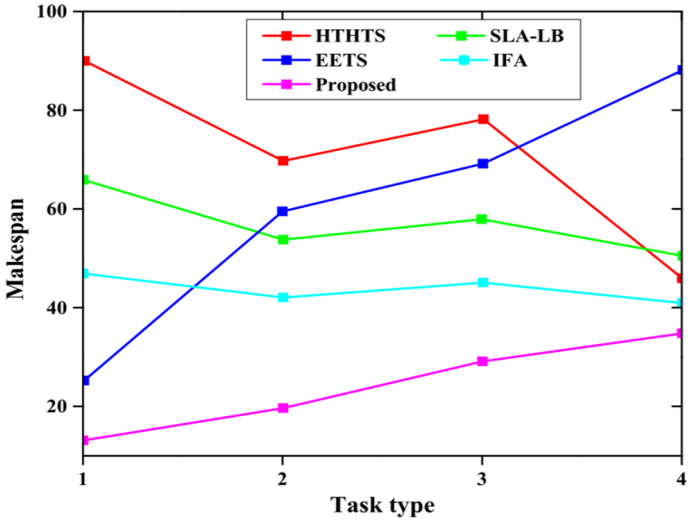

Fig. 3 State-of-art comparison of response time based on task type

approaches such as HTHTS, SLA-LB, EETS, and IFA, the proposed method obtains minimum makespan values, as shown in Fig. 4.

$$FS_{j,k} = \frac{SL_k}{C_j} \tag{30}$$

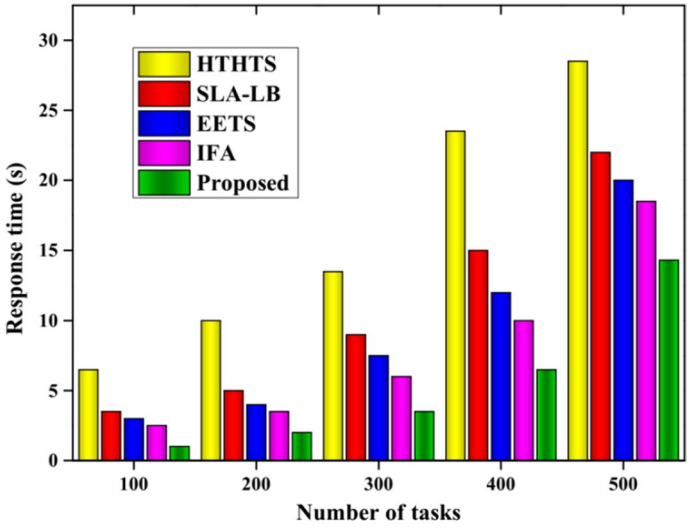

Fig. 4 State-of-art comparison of response time based on the number of tasks

$$PS_{j,k} = CS_{(k-1)j} \tag{31}$$

The state-of-art comparison of response time with respect to arrival rate is described in Fig. 5. The existing Hybrid Tabu–Harmony task scheduling (HTHTS) [9], Service level agreement-based Load Balancing (SLA-LB) [10], Energy-efficient task-scheduling (EETS) [12] and Improved Firework Algorithm (IFA) [13] with the proposed method validate the performance of proposed work. With an average response time of 200 to 1000 s, this test uses a variable arrival rate ranging from 20 to 100. The proposed method takes less average response time than other conventional methods including HTHTS, SLA-LB, EETS, and IFA.

6.3 Fault tolerance Performance

The two most critical Fault Tolerance tasks are fault recovery and fault detection. To perform the former in a decentralized manner, all hosts place a monitoring ring on top of a peer ring and observe their counter-clockwise neighbors. The backup host will have all failed node tuples, which will also validate the given status. Backup nodes are never given additional resources; they are only utilized for recovery. The backup node gets overburdened as a result of or during the takeover. The offload initiates the scale-out or adaptive load balancing procedure without interfering with routine operations. The failure rate of different techniques is identified and the results are presented in Fig. 6a and b in terms of network usage and delay. The network usage is computed in terms of kilobytes and the delay is computed in terms of milliseconds. For a failure rate of 0.1%, the proposed model offers better performance in both delay and network usage.

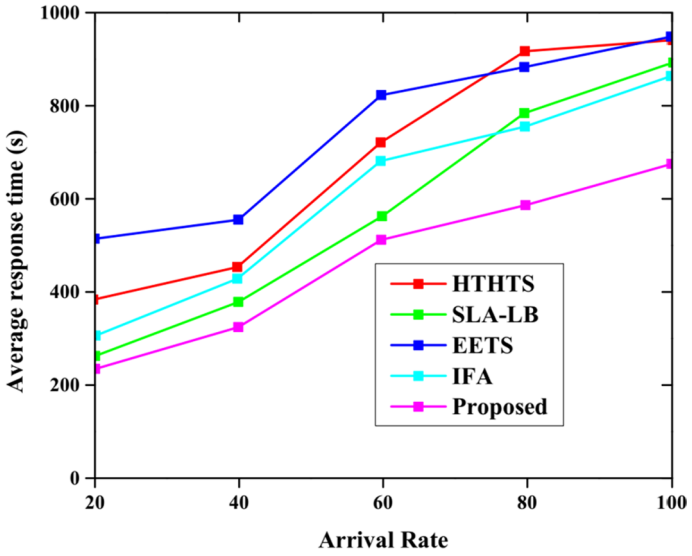

Fig. 5 State-of-art comparison of response time with respect to the arrival rate

6.4 Performance Efficiency

The efficiency of the different models taken for comparison is mainly computed using the prediction time and prediction accuracy. The prediction time primarily computes the time required to allocate the appropriate task to the intended usage within the user-specified time limit. The prediction accuracy mainly implies the total amount of tasks in the queue that are executed within the limited time specified by the user to the overall tasks present in the cloud. Hence an efficient methodology should consume minimal prediction time and higher prediction accuracy. The predicted task state is measured using a minimum and maximum task arrival state threshold value. The state-of-art comparison of overall efficiency is delineated in Fig. 7. This analysis is conducted in the context of user requests arrival rate and time. The proposed technique is compared to existing Hybrid Tabu-Harmony task scheduling (HTHTS) [9], Service level agreement-based Load Balancing (SLA-LB) [10], Energy-efficient task scheduling (EETS) [12], and Improved Firework Algorithm (IFA) [13]. The proposed method demonstrated superior efficiency results to other existing methods including HTHTS, SLA-LB, EETS, and IFA.

6.5 Load Balancing Performance

The load balancing capability of any strategy is determined by its ability to maximize resource utilization while minimizing total completion time. A system with a higher load balancing capacity has a higher task configuration capacity by identifying the optimal VM for task allocation. In this way, the resource utilization capability is increased when the number of incoming tasks with different sizes is managed correctly by the VM assigned. The load balancing performance results are delineated in Fig. 8. The load balancing test checks the task scheduling performances provided. Figures 8a–c depict the load balancing analysis in terms of CPU consumption, I/O utilization, and memory utilization. The proposed method is used to compare load balancing performance with existing Hybrid Tabu-Harmony task scheduling (HTHTS) [9], Service level agreement-based Load Balancing (SLA-LB) [10], Energy-efficient task scheduling (EETS) [12], and Improved Firework Algorithm (IFA) [13]. The HTHTS, on the other hand, selected idle resources

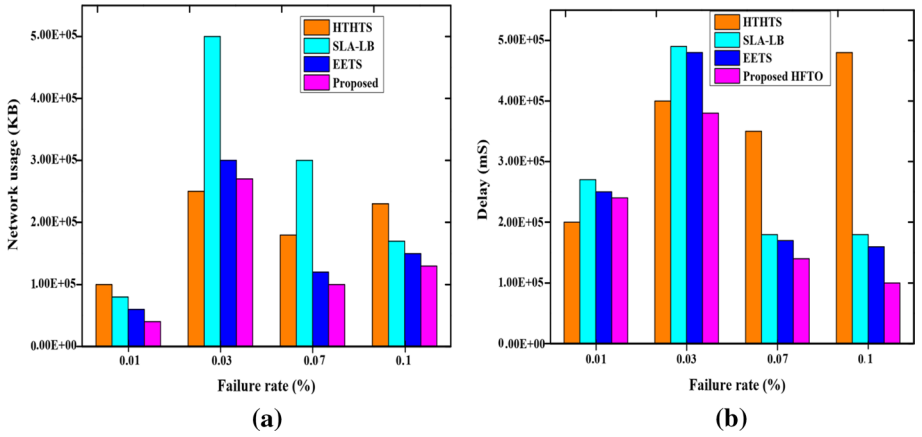

Fig. 6 Impact of failure rate. **a** Network usage and **b** Delay

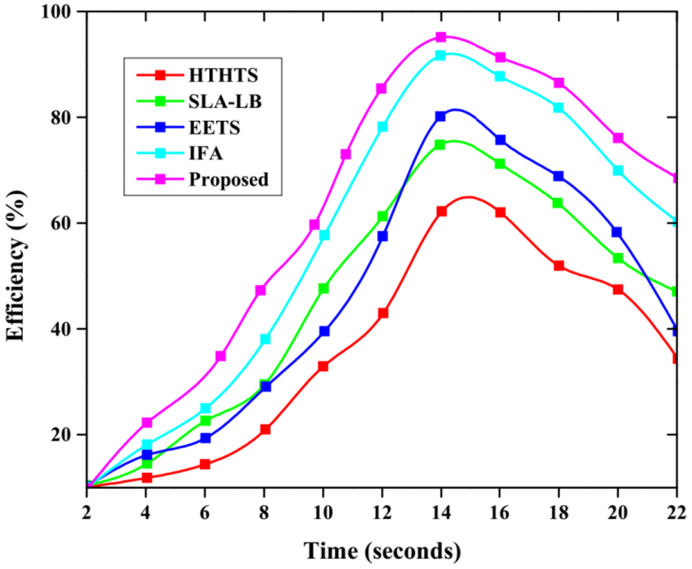

Fig. 7 State-of-art comparison of efficiency

over SLA-LB, EETS, and IFA. However, the proposed work allocates resources more efficiently than existing methods. Based on the investigation of Fig. 8a–c, the proposed techniques take less CPU utilization, Input–Output utilization (IOU), and memory utilization. We mainly took the terms such as CPU utilization, IOU, and memory into consideration because they are mainly related to resource wastage. An effective load balancing scheme should resist resource wastage as maximum as possible.

7 Conclusion

In this article, a task scheduling framework using Hybrid Firebug and Tunicate Optimization (HFTO) technique is presented. CloudSim simulates the proposed model's performance. The state-of-art comparison is carried out using the proposed method with existing Hybrid Tabu–Harmony task scheduling (HTHTS), Service level agreement-based Load Balancing (SLA-LB), Energy-efficient task-scheduling (EETS), and Improved Firework Algorithm (IFA). Computationally intensive tasks are assigned to resources with low CPU utilisation, whereas lightweight tasks are assigned to resources with high CPU utilisation. The HFTO algorithm improves the Quality of Service (QoS) factors fault tolerance, response time, efficiency, and makespan. In terms of cloud task scheduling performance and load balancing efficiency, the proposed strategy outperformed the previous techniques.

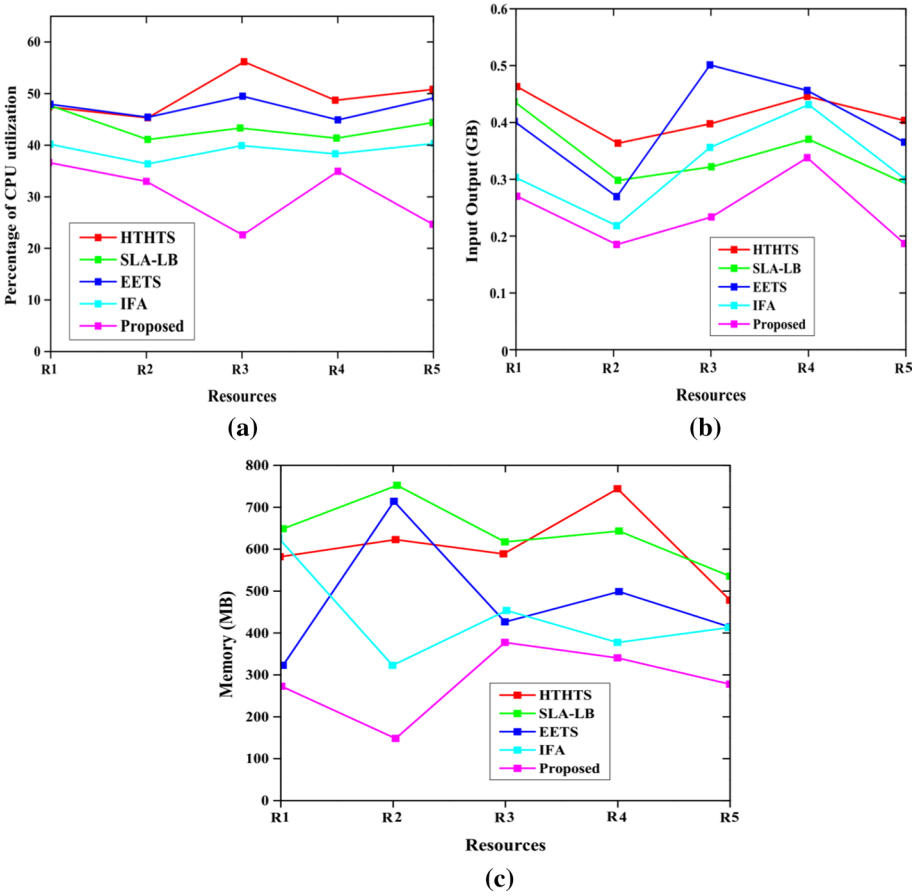

Fig. 8 Load balancing performances, **a** CPU utilization, **b** Input Output utilization, and **c** Memory utilization

Author Contributions All authors agreed on the content of the study. MN,GN and PK collected all the data for analysis. MN agreed on the methodology. MN,GN and PK completed the analysis based on agreed steps. Results and conclusions arediscussed and written together. The author read and approved the final manuscript.

Funding Not applicable.

Data Availability and Materials Data sharing is not applicable to this article as no new data were created or analyzed in this study.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Human and Animal Rights This article does not contain any studies with human or animal subjects performed by any of the authors.

Informed Consent Informed consent was obtained from all individual participants included in the study.

References

1. Geng, X., Yu, L., Bao, J., & Fu, G. (2019). A task scheduling algorithm based on priority list and task duplication in cloud computing environment. *Web Intelligence, 17*(2), 121–129.
2. Ramasubbareddy, S., Swetha, E., Luhach, A. K., & Srinivas, T. A. (2021). A multi-objective genetic algorithm-based resource scheduling in mobile cloud computing. *International Journal of Cognitive Informatics and Natural Intelligence (IJCINI), 15*(3), 58–73. https://doi.org/10.4018/IJCINI.20210701.oa5
3. Arulkumar, V., & Bhalaji, N. (2021). Performance analysis of nature inspired load balancing algorithm in cloud environment. *Journal of Ambient Intelligence and Humanized Computing, 12*(3), 3735–3742.
4. Yiqiu, F., Xia, X., and Junwei, G., (2019). Cloud computing task scheduling algorithm based on improved genetic algorithm. *In 2019 IEEE 3rd information technology, networking, electronic and automation control conference (ITNEC)* (pp. 852–856). IEEE.
5. Houssein, E. H., Gad, A. G., Wazery, Y. M., & Suganthan, P. N. (2021). Task scheduling in cloud computing based on meta-heuristics: Review, taxonomy, open challenges, and future trends. *Swarm and Evolutionary Computation, 62*, 100841.
6. Gupta, A., & Garg, R. (2017). Load balancing based task scheduling with ACO in cloud computing. In 2017 *International conference on computer and applications (ICCA)* (pp. 174–179). IEEE.
7. Fang, Y., Wang, F., & Ge, J. (2010). A task scheduling algorithm based on load balancing in cloud computing. *International conference on web information systems and mining* (pp. 271–277). Springer.
8. Ebadifard, F., & Babamir, S. M. (2021). Autonomic task scheduling algorithm for dynamic workloads through a load balancing technique for the cloud-computing environment. *Cluster Computing, 24*(2), 1075–1101.
9. Alazzam, H., Alhenawi, E., & Al-Sayyed, R. (2019). A hybrid job scheduling algorithm based on Tabu and Harmony search algorithms. *The Journal of Supercomputing, 75*(12), 7994–8011.
10. Lavanya, M., Shanthi, B., & Saravanan, S. (2020). Multi objective task scheduling algorithm based on SLA and processing time suitable for cloud environment. *Computer Communications, 151*, 183–195.
11. Lu, Y., & Sun, N. (2019). An effective task scheduling algorithm based on dynamic energy management and efficient resource utilization in green cloud computing environment. *Cluster Computing, 22*(1), 513–520.
12. Khorsand, R., & Ramezanpour, M. (2020). An energy-efficient task-scheduling algorithm based on a multi-criteria decision-making method in cloud computing. *International Journal of Communication Systems, 33*(9), e4379.
13. Wang, S., Zhao, T., & Pang, S. (2020). Task scheduling algorithm based on improved firework algorithm in fog computing. *IEEE Access, 8*, 32385–32394.
14. Noel, M. M., Muthiah-Nakarajan, V., Amali, G. B., & Trivedi, A. S. (2021). A new biologically inspired global optimization algorithm based on firebug reproductive swarming behaviour. *Expert Systems with Applications, 183*, 115408.
15. Shyaamini, B., & Senthilkumar, M. (2006). Multi objective particle swarm optimization for performance testing in web application. *ARPN Journal of Engineering and Applied Sciences, 13*(11), 1–9.
16. Houssein, E. H., Helmy, B. E. D., Elngar, A. A., Abdelminaam, D. S., & Shaban, H. (2021). An improved tunicate swarm algorithm for global optimization and image segmentation. *IEEE Access, 9*, 56066–56092.
17. Ergu, D., Kou, G., Peng, Y., Shi, Y., & Shi, Y. (2013). The analytic hierarchy process: Task scheduling and resource allocation in cloud computing environment. *The Journal of Supercomputing, 64*(3), 835–848.
18. Dhinesh Babu, L. D., & Krishna, P. V. (2013). Honey bee behavior inspired load balancing of tasks in cloud computing environments. *Applied Soft Computing, 13*(5), 2292–2303.
19. Xu, X., Fu, S., Cai, Q., Tian, W., Liu, W., Dou, W., & Liu, A. X. (2018). Dynamic resource allocation for load balancing in fog environment. *Wireless Communications and Mobile Computing, 2018*, 1–15.
20. Polepally, V., & Chatrapati, K. S. (2019). Dragonfly optimization and constraint measure-based load balancing in cloud computing. *Cluster Computing, 22*(1), 1099–1111.
21. Xingjun, L., Zhiwei, S., Hongping, C., & Mohammed, B. O. (2020). A new fuzzy-based method for load balancing in the cloud-based Internet of things using a grey wolf optimization algorithm. *International Journal of Communication Systems, 33*(8), e4370.

22. Muthsamy, G., & Ravi Chandran, S. (2020). Task scheduling using artificial bee foraging optimization for load balancing in cloud data centers. *Computer Applications in Engineering Education, 28*(4), 769–778.
23. Deng, Z., Cao, D., Shen, H., Yan, Z., & Huang, H. (2021). Reliability-aware task scheduling for energy efficiency on heterogeneous multiprocessor systems. *The Journal of Supercomputing, 77*(10), 11643–11681.
24. Luppold, A., Oehlert, D., & Falk, H. (2020). Compiling for the worst case: Memory allocation for multi-task and multi-core hard real-time systems. *ACM Transactions on Embedded Computing Systems (TECS), 19*(2), 1–26.
25. Roy, A., & Livny, M. (2004). *Grid resource management state of the art and future trends* (pp. 135–144). Springer.
26. Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., & Buyya, R. (2011). CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience, 41*(1), 23–50.
27. Boveiri, H. R., Khayami, R., Elhoseny, M., & Gunasekaran, M. (2019). An efficient Swarm-Intelligence approach for task scheduling in cloud-based internet of things applications. *Journal of Ambient Intelligence and Humanized Computing, 10*(9), 3469–3479.
28. Abd Elaziz, M., Xiong, S., Jayasena, K. P. N., & Li, L. (2019). Task scheduling in cloud computing based on hybrid moth search algorithm and differential evolution. *Knowledge-Based Systems, 169*, 39–52.
29. Houssein, E. H., Gad, A. G., Wazery, Y. M., & Suganthan, P. N. (2021). Task scheduling in cloud computing based on meta-heuristics: Review, taxonomy, open challenges, and future trends. *Swarm and Evolutionary Computation, 2021*, 100841.
30. Prem Jacob, T., & Pradeep, K. (2019). A multi-objective optimal task scheduling in cloud environment using cuckoo particle swarm optimization. *Wireless Personal Communications, 109*(1), 315–331.
31. Sreenu, K., & Sreelatha, M. (2019). W-Scheduler: Whale optimization for task scheduling in cloud computing. *Cluster Computing, 22*(1), 1087–1098.
32. Mapetu, J. P. B., Chen, Z., & Kong, L. (2019). Low-time complexity and low-cost binary particle swarm optimization algorithm for task scheduling and load balancing in cloud computing. *Applied Intelligence, 49*(9), 3308–3330.
33. Prassanna, J., & Venkataraman, N. (2019). Adaptive regressive holt–winters workload prediction and firefly optimized lottery scheduling for load balancing in cloud. *Wireless Networks, 2019*, 1–19.
34. Sundararaj, V., 2019. Optimal task assignment in mobile cloud computing by queue based ant-bee algorithm. Wireless Personal Communications, 104(1), pp.173–197. 10.1007/s11277-018-6014-9
35. Manikandan, N., Gobalakrishnan, N. and Pradeep, K., 2022. Bee optimization based random double adaptive whale optimization model for task scheduling in cloud computing environment. Computer Communications, 187, pp.35–44. 10.1016/j.comcom.2022.01.016

Dr. Manikandan Nanjappan obtained his Bachelors of Engineering degree in Computer Science Engineering from Anna University 2005. Then he obtained his Master's of Engineering degree in in Computer Science Engineering from Anna University 2011. He completed his PhD in Sathyabama Institute of Science and Technology 2020 and presently he is an Assistant Professor of Department of Data science and Business Systems, SRM Institute of Science and Technology, SRM Nagar, Kattankulathur, Chennai, TN, India. His specializations include cloud computing, Operating systems, networking.

Gobalakrishnan Natesan pursued his Bachelor's degree in Computer Science and Engineering at Madras University, Tamilnadu, India in 2004. He then obtained his Master's degree in Computer Science and Engineering from Anna University, Tamilnadu, India in 2011. He completed his Ph.D in Sathyabama Institute of Science and Technology, Chennai, India and working an Assistant Professor in the School of Computer Science and Engineering, VIT University, Chennai, Tamilnadu, India. His current research interests are Cloud computing, Virtualization and Big Data.

Pradeep Krishnadoss pursued his Bachelor's degree in Information Technology at Anna University, Tamilnadu, India in 2005. He then obtained his Master's degree in Software Engineering from Bharathidhasan University, Tamilnadu, India in 2008. He received his Ph.D from Sathyabama Institute of Science and Technology, Chennai, India and working as Associate Professor in the Department of Information Technology, Sri Venkateswara College of Engineering, Chennai, Tamilnadu, India. His current research interests are Cloud computing, IoT and Big Data. He is a life time member of ISTE.

Authors and Affiliations

Manikandan Nanjappan[1] · Gobalakrishnan Natesan[2] · Pradeep Krishnadoss[3]

[1] Department of Data science and Business Systems, School of Computing, College of Engineering and Technology, Faculty of Engineering and Technology, SRM Institute of Science and Technology, SRM Nagar, Kattankulathur, Chennai, Tamilnadu, India

[2] Department of Information Technology, Sri Venkateswara College of Engineering, Chennai, Tamilnadu, India

[3] School of Computer Science and Engineering, Vellore Institute of Technology, Chennai, Tamilnadu, India