Load Balancing in the Cloud: A Comparative Evaluation of the Red Deer Algorithm

Krish Sharma[1], Tanmay Tushar[1], Inderjeet Kaur[2], Ashish Jain[1*]

[1]Department of Information Technology, Manipal University Jaipur, Jaipur, 303007, Rajasthan, India.
[2]Department of Computer Science and Engineering, Ajay Kumar Garg Engineering College, Ghaziabad, Uttar Pradesh, India.

*Corresponding author(s). E-mail(s): ashish.jain@jaipur.manipal.edu;

Abstract

This paper investigates the critical challenge of load balancing in cloud computing, a key factor influencing the performance, reliability, and efficiency of cloud-based services. With the rise of cloud computing technologies, the need for effective workload distribution across heterogeneous and geographically distributed resources has become increasingly complex. Load balancing is categorized as an NP-hard problem due to its vast solution space, necessitating the use of metaheuristic algorithms for efficient problem-solving. In this study, we propose a load balancing approach based on Red Deer Algorithm (RDA). Additionally, a comparative performance analysis is conducted between various metaheuristic load balancing algorithms to evaluate their effectiveness based on metrics such as makespan time, response time, resource utilization. Simulation results using CloudSim demonstrate that the RDA-based method significantly reduces costs and response time. The findings support the potential of metaheuristic techniques to provide near-optimal solutions within reasonable timeframes, thereby enhancing Quality of Service (QoS) and resource utilization in cloud environments.

Keywords: Metaheuristic Algorithms, Red Deer Algorithm, Cloud Computing, Grey Wolf Optimization, Genetic Algorithm

1 Introduction

Cloud computing is a model to deliver resources such as servers, storage systems, databases, network facilities, software applications, and analysis tools over the Internet, which is also known as "the cloud" [1]. The use of cloud computing and its services enables users to access applications from any point on the planet. This technique is very different from other hosting due to two core elements, i.e., the Cloud Service Provider and Cloud Service User [2]. It is often the case that a service incurs a charge. For the cloud computing service customers get to use the "pay as you go" and "on-demand" model where they are not bound by long-term contracts. The self-service feature allows data to be provisioned automatically without the need for user input, sales orders, new service setup, as well as long and complicated contractual agreements, thus enhancing customer service and ease of use. This implies that the acquisition of cloud services is fully automated and plays a significant role in offering these services at a competitive price [3]. Due to the dynamic and distributed nature of cloud computing environments, where resources are dynamically provisioned and consumed over the Internet as needed, proper management of the resources is crucial. As the number of applications and users that depend on cloud services continues to increase on a daily basis, it is crucial to make sure that workloads are properly distributed among accessible computing nodes to keep the system responsive, reliable, and performing. This is where load balancing is needed [4].

Load balancing is the efficient and balanced assignment of tasks to computing resources such as virtual machines (VMs) to maintain resource utilization and load satisfaction (i.e., processor load, main and secondary memory used, delays, or networking load) of users and increase the rate of resource productivity [5]. When some VMs have higher load volumes than others, the efficiency of the system is reduced. Load balancing addresses this challenge by implementing an intelligent and equitable distribution of tasks across available computing resources [6]. Load Balancing aims to maintain acceptable levels of resource utilization, measured by processor activity, memory consumption, response times, or network traffic, while simultaneously satisfying user requirements and maximizing resource efficiency [7]. A balanced load balancing mechanism considers both current resource availability and specific task demands to make optimal allocation decisions, thereby reducing incoming traffic bottlenecks, increasing performance, enhancing system throughput, and ensuring cloud infrastructure is utilized to its full potential. That is why the system through which load balancing is performed is of paramount importance [8]. Deterministic or rule-based algorithms will be found wanting in dealing with the complexity, size, and uncertainty of the cloud environment. Metaheuristic algorithms, however, have been found to be promising options to resolve load balancing issues because they are capable of efficiently navigating large solution spaces and identifying near-optimal solutions within acceptable computation times [9].

A metaheuristic solution can be defined as an interactive set of processes that govern the exploration strategy and the exploitation of the search landscape [10, 11]. Metaheuristic methods are one of the strategies that can be applied to resolve performance issues, for example, task scheduling [12, 13]. Metaheuristic algorithms are high-level problem-independent methods that guide lower-level heuristics to finding

optimal or near-optimal solutions to complicated optimization problems. These algorithms, e.g., Genetic Algorithms (GA), Particle Swarm Optimization (PSO) [14], and Ant Colony Optimization (ACO) [15], are inspired by natural phenomena like evolutionary processes, collective behavior, or reproduction strategies of certain animal species [10, 16]. Even though they do not guarantee the achievement of a global optimum, such approaches are well adapted to providing acceptable solutions for NP-hard problems, especially if exhaustive search is computationally infeasible. Their flexibility, scalability, and adaptability make them particularly well adapted to dynamic environments like cloud computing, where conditions change constantly and optimum task-resource assignment needs constant re-tuning [17].

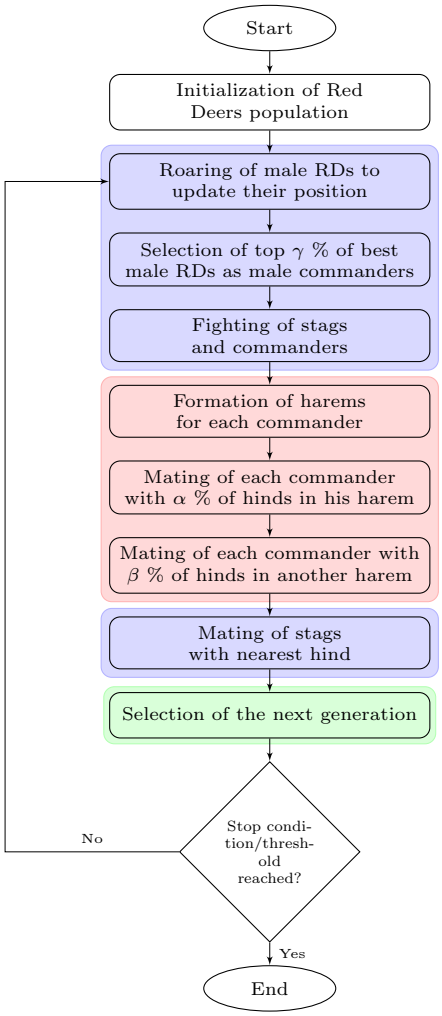

Fig. 1 Flow Chart of RDA [18]

Currently, the selection of the most suitable services has been proposed to achieve optimally by QoS, addressing limitations related to connectivity and the definition of QoS criteria [19]. QoS remains one of the most critical aspects in addressing challenges within cloud computing. In the present study, a novel load balancing technique is introduced based on the RDA, integrated with QoS considerations to identify the best possible solution. RDA draws inspiration from the natural mating behavior of red deer [18]. The algorithm begins by initializing a population of red deer, which are then classified into male and female individuals. All male red deer update their positions and then mate. After all mating operations, a new generation is formed by selecting offspring based on fitness, and the process iterates until a predefined number of generations is reached [18]. This biologically inspired structure allows RDA to effectively balance exploration and exploitation, making it highly suitable for solving complex optimization problems such as load balancing in cloud computing, where dynamic resource allocation and efficiency are essential [20].

The structure of rest of this paper follows as: Section 2 discusses related work in the domain. Section 3 presents the proposed model. The development of the RDA in accordance with the proposed model is detailed in Section 4. Section 5 outlines the experimental setup, while Section 6 presents the results and analysis. Finally, Section 7 concludes the paper with future research directions.

2 Related Work

Kruekaew and Kimpan [21] suggested the Heuristic Task Scheduling with Artificial Bee Colony (HABC) algorithm, as well as a non-concentrated Artificial Bee Colony (ABC) load-balancing strategy, to improve virtual machine scheduling and load allocation in cloud computing environments. HABC integrates swarm intelligence from ABC with heuristic scheduling, while the non-concentrated ABC draws inspiration from honey-bee behavior to dynamically manage load balancing. HABC was compared against ACO, PSO, and Improved PSO (IPSO). HABC excelled in minimizing makespan and enhancing load balancing across homogeneous and heterogeneous setups. The non-concentrated ABC approach further reduced makespan and response time by dynamically adapting to demand fluctuations but also showed drawbacks, including lower reliability and higher operational costs. However, both methods require careful parameter tuning, as performance depends heavily on problem size and dataset characteristics.

Kokilavani and Amalarethinam [22] introduced a task scheduling algorithm designed to efficiently allocate tasks to available resources. The algorithm operates iteratively, focusing on minimizing the finish time of tasks. The authors presented results indicating that this algorithm effectively reduces energy consumption and overload. However, they also acknowledged that the algorithm's design does not explicitly address factors such as the reliability of the resources or the response time experienced by the tasks.

Goyal and Singh [23] devised an innovative approach to balance workloads in grid computing with their Ant-based Load Balancing Algorithm(ALBA). Unlike traditional ACO, where ants leave pheromone trails on paths, ALBA marks resources themselves

Table 1 Comparison of related load balancing algorithms in cloud computing

Author(s)	Algorithm	Advantages	Limitations
Kruekaew and Kimpan [21]	HABC, ABC	Minimizes makespan; improves load balance	Needs parameter tuning; high operational cost
Kokilavani and Amalarethinam [22]	Heuristic Scheduling	Reduces energy consumption and overload	Ignores reliability and response time
Goyal and Singh [23]	ALBA	Better load distribution in grid/cloud	High response time and cost
Makasarwala and Hazari [24]	GA	Optimizes response time	Ignores resource needs, lacks robustness
Sefati et al. [25]	GWO	Lower response time and cost; considers reliability	Computational overhead; lacks security/scalability
Devaraj et al. [26]	PSO, Firefly	Energy efficient load balancing	Complex hybrid algorithm
Lilhore et al. [27]	PSO, Firefly	Cloud performance evaluation	

to indicate how busy or free they are based on task updates. They tested ALBA using a simulation tool, varying the number of tasks and resources, and found it distributed work more evenly than older methods. They also applied a similar concept to cloud computing, leveraging complex network principles to enhance load balancing in dynamic, diverse systems. However, it has a downside: it tends to have high response times and costs, which could pose challenges for some applications.

Makasarwala and Hazari [24] propose a GA for load balancing in cloud computing, initializing a population of solutions where request priority, based on job length (time), guides task assignment to VMs. Using permutation encoding with decimal numbers for chromosome representation, the GA evolves through a fitness function that selects well-balanced solutions for crossover and mutation, optimizing response times. Simulations show that it outperforms traditional methods. However, its focus on time-based priority overlooks other factors like resource needs, and it lacks robustness in reliability and availability, with evaluations limited to response time.

Sefati et al. [25] proposed a load-balancing method for cloud computing environments using the Grey Wolf Optimization (GWO) algorithm to efficiently distribute workloads across VMs while considering resource reliability to enhance user satisfaction and system productivity. The method employs the GWO algorithm, inspired by the social behavior of grey wolves, to select a cluster head, calculate VM load thresholds, and prioritize underloaded VMs with higher QoS for task allocation, demonstrating reduced costs and response times compared to other algorithms in

CloudSim simulations. However, the approach has potential drawbacks, including the computational overhead of the GWO algorithm, which necessitates a trade-off analysis between solution quality and computational cost, and while the research emphasizes load balancing and QoS, it could be strengthened by incorporating other cloud computing factors such as security, scalability, and fault tolerance; the authors suggest future research directions including the integration of machine learning and multi-criteria decision-making methods and the exploration of dynamic load balancing for dependent tasks to address these limitations and develop more robust solutions.

Devaraj et al. [26] suggesterd a new load balancing algorithm, Firefly and Improved Multi-Objective Particle Swarm Optimization (FIMPSO) which is for cloud computing. It's a hybrid of the Firefly Algorithm (FF) and Improved Multi-Objective Particle Swarm Optimization (IMPSO). They claim that their FIMPSO model outperformed in terms of good average load acquisition, better resource utilization (CPU and memory), faster task responses, better makespan, and throughput compared to other approaches. Although the algorithm demonstrates significant improvements in energy efficiency and other performance attributes, the results also showed a reliability score of 67%. This indicates that parameters like fault handling or achieving higher task completion rates under varying conditions may require more attention.

Lilhore et al. [27] developed a cloud computing load balancing model using a combination of improved Particle Swarm Optimization (MPSO) and improved Firefly Algorithm (IMFA). This hybrid process results in improved load balancing, lower average response time, faster data center processing time, and less waiting time for tasks. The authors provided simulation results illustrating that their MPSO-IMFA approach performed better than traditional PSO and other load balancing approaches in these aspects, which means better resource utilization and increased system throughput. While the algorithm demonstrates improvements in performance and efficiency through enhanced task scheduling, the description is geared towards these results without detailing dynamic resource management or the specific energy usage of the balancing mechanism.

3 Proposed Model

In cloud computing, vast networks of data centers and users are spread globally. When numerous user requests flood the cloud simultaneously, efficient organization and service delivery are essential. The cloud must distribute workloads evenly across resources to ensure fairness. Load balancing remains a key challenge, requiring equal distribution of tasks to maintain user satisfaction. Various load-balancing algorithms exist for distributed, grid, and cloud systems [10]. This study introduces a novel method for load balancing based on QoS to minimize response time and resource allocation costs. The proposed approach employs the RDA algorithm to achieve effective load balancing in a cloud environment.

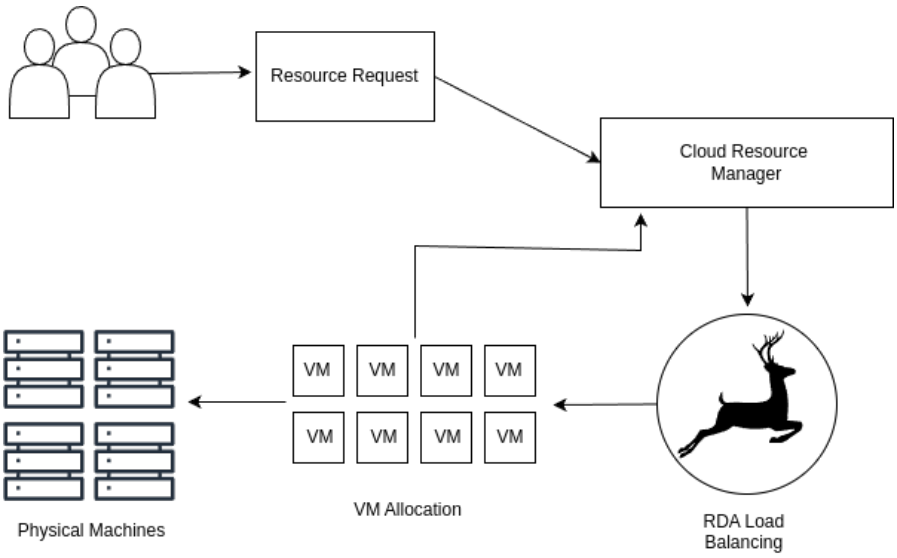

Fig. 2 The proposed load-balancing system

3.1 Problem Statement

Consider cloud C, which consists of N physical machines, each containing M VMs.

$$C = \{PM_1, PM_2, \ldots, PM_N\} \tag{1}$$

Additionally, every physical machine consists of multiple VMs configured as follows:

$$PM_i = \{VM_{i1}, VM_{i2}, \ldots, VM_{iM}\} \tag{2}$$

VM_1 represents the first virtual machine, and VM_M denotes the final one. Our aim is to minimize costs and maximize the utilization of resources by evenly distributing workload across all the VMs. Efficient load balancing is the key to our approach; otherwise, the system would waste unnecessary time and energy on its tasks. To tackle this, our study proposes an efficient, multi-objective load-balancing approach using the RDA algorithm.

Our model treats the resource allocation as a multi-objective optimization problem and seeks to:

1. Minimize the makespan time to enhance task completion time and efficiency,
2. Maximize utilization of resources to improve infrastructure efficiency,
3. Minimize costs to make it economically viable,
4. Reduce response time to maintain QoS.

These objectives tend to be conflicting with one another, and they need to be balanced properly. For example, aggressively reducing makespan may raise the demand for provisioning more resources, which can raise costs and lower overall utilization [25].

The model employs a weighted approach towards these objectives, allowing system administrators to assign weights to some measurements based on their intended use. This renders it adaptable to evolving requirements while maintaining the overall system performance.

3.2 Parameters of QoS

Makespan, utilization of resources, response time, and cost-effectiveness are performance measures that, collectively, constitute a holistic framework for quantifying the effectiveness of resource allocation policies [21]. The model introduced here attempts to promote resource allocation in cloud environments by balancing these performance measures.

3.2.1 Makespan

Makespan time represents the total time required to complete the scheduled tasks [25]. In our model, we are considering the maximum makespan time among all VMs, it is given as following:

$$\text{Makespan} = \max\{CT_1, CT_2, \ldots, CT_M\} \tag{3}$$

Where CT(i) represents the completion time of virtual machine i, and M is the total number of available VMs. Our goal is to reduce makespan time to guarantee timely execution of tasks.

3.2.2 Resource Utilization

Resource utilization is the ratio of time a VM is actively processing tasks to the makespan time [25]. Resource utilization for an individual VM:

$$\text{Utilization}_{VM_j} = \frac{\sum_{i=1}^{n} PT_{ij}}{\text{Makespan}} \tag{4}$$

Where PT_{ij} represents the processing time of task i on VM_j. The total resource utilization across the cloud is then determined by:

$$\text{Resource Utilization} = \frac{\sum_{j=1}^{m} \text{Utilization}_{VM_j}}{m} \tag{5}$$

This metric provides information on how effectively available computational resources are being used in the present situation. Increased utilization ratios indicate better resource management, which can reduce the operational expenses as well as the environmental impact.

3.2.3 Response Time (RT)

We model the response time in our model by utilizing SpaceShared basis queuing behavior of tasks on Vms. SpaceShared is a resource allocation policy where each cloudlet uses a single core at a time, and the remaining wait in a queue until other resources become available to be utilized [28].

The response time is defined as the time that a cloudlet takes from the time of arrival to its completion time. In our model, we leveraged the SpaceShared model to simulate the queuing behavior of tasks on Vms. The SpaceShared model of allocation is a resource allocation policy where each cloudlet can get only a single core at a time, while others must wait in a queue until other resources become available [28].

The execution time for cloudlet C_i on VM v_j is:

$$RT_{exec}(C_i, v_j) = \frac{Len}{MIPS_j} \tag{6}$$

Where Len is the cloudlet length in Million Instructions (MI), and $MIPS_j$ is the processing speed of VM v_j.

Cloudlets on the same VM execute sequentially, so $RT(C_i)$ includes the time of all previous cloudlets. The average response time is:

$$RT_{avg} = \frac{1}{n} \sum_{i=1}^{n} RT(C_i) \tag{7}$$

Lower values indicate better scheduling and faster execution.

3.2.4 Cost

The cost model incorporates both resource consumption and temporal factors [25]. For a set of K VMs assigned to user requests, the total cost is expressed as:

$$\text{Cost} = \sum_{i=1}^{K} C_i \cdot T_i \tag{8}$$

where C_i represents the cost rate of VM_i, and T_i is the duration for which a user utilizes VM_i. This formulation enables fine-grained cost analysis based on actual resource consumption patterns. It accounts for varying computational requirements across different tasks, ensuring users pay for actual usage rather than static allocations.

3.3 Fitness Function

All the QoS parameters use different units, Some parameters (e.g., utilization) have a positive contribution to fitness, while others (e.g., response time, cost, makespan) have a negative impact [25]. Hence, they need to be normalised on a single scale to measure the objective function.

Equation 9 and 10 represents the normalization formula for the maximum and minimum parameters, respectively [29].

$$N_{CS,Q^i} = \begin{cases} \frac{Q^i_{\max} - CS \cdot Q^i}{Q^i_{\max} - Q^i_{\min}}, & Q^i_{\max} \neq Q^i_{\min} \\ 1, & \text{otherwise} \end{cases} \tag{9}$$

$$N_{CS,Q^i} = \begin{cases} \frac{CS \cdot Q^i - Q^i_{\min}}{Q^i_{\max} - Q^i_{\min}}, & Q^i_{\max} \neq Q^i_{\min} \\ 1, & \text{otherwise} \end{cases} \tag{10}$$

The fitness function is given as Algorithm 1:

Algorithm 1 Fitness Function

```
1: procedure EvaluateFitness(solution)
2:     Initialize workload, cost, responseTime, utilization
3:     Randomly set VM MIPS and cost
4:     for each cloudlet do
5:         Assign cloudlet to VM based on solution
6:         Update workload and VM task list
7:     end for
8:     for each VM do
9:         for each cloudlet in VM do
10:            Calculate execTime
11:            Accumulate processingTime, cost, responseTime
12:        end for
13:    end for
14:    Calculate makespan, utilization, avgResponseTime
15:    Normalize all QoS metrics
16:    Compute and return weightedFitness
17: end procedure
```

To effectively balance the QoS parameters during resource allocation, a unified fitness function is designed which evaluates each candidate solution based on normalized QoS metrics, ensuring correctness regardless of their original scales [25].

The overall fitness F of a solution is computed as:

$$Fitness = w_1 \cdot N_{makespan} + w_2 \cdot N_{utilization} + w_3 \cdot N_{cost} + w_4 \cdot N_{RT} \tag{11}$$

where $N_{makespan}$ 3, $N_{utilization}$ 4, 5, N_{cost} 8, and N_{RT} 7 are the normalized parameters, and w_1 to w_4 are user-defined weights satisfying $\sum w_i = 1$. Optimizing solely one parameter could lead to resource under utilization. Hence, By assigning appropriate weights, users could emphasize on specific objectives based on their operational context.

4 Development of the RDA According to the Proposed Model

In the proposed approach, the primary objective is to support real-time cloud applications, where adherence to task deadlines is critical. Each task is associated with a user-defined execution time constraint, and the system endeavors to complete the execution of the task within the specified timeframe to uphold QoS requirements.

The RDA is a nature-inspired metaheuristic technique modeled after the mating behavior of Scottish red deer during the breeding season. In the context of cloud computing, this algorithm is adapted to solve the resource allocation problem, specifically mapping cloudlets to VMs by simulating the social dynamics of red deer. Each red deer represents a candidate solution encoding a potential mapping of cloudlets to VMs.

Algorithm 2 RDA for Load Balancing in Cloud Computing

procedure RedDeerAlgorithm($numVMs, numCloudlets$)
  Initialize: $populationSize, numMales, numHinds, \alpha, \beta, \gamma, UB, LB, maxIterations$
  $numCommanders \leftarrow \lfloor \gamma \times numMales \rfloor$
  $numStags \leftarrow numMales - numCommanders$
  $bestFitness \leftarrow \infty$
  Initialize Population: Generate $populationSize$ random individuals
  for all deer d in $population$ **do**
    $d.fitness \leftarrow 1$ EvaluateFitness($d.position$)
  end for
  Sort population by fitness
  $males \leftarrow$ top $numMales$ individuals
  $hinds \leftarrow$ remaining individuals
  $commanders \leftarrow$ top $numCommanders$ from $males$
  $stags \leftarrow$ remaining $males$
  for $iter = 1$ to $maxIterations$ **do** ▷ Roaring Phase
    for all $male$ in $males$ **do**
      Update $male$ position if fitness improves via roaring
    end for
    Sort $males$ by fitness
    Update $commanders$ and $stags$
    ▷ Fighting Phase
    for all $commander$ in $commanders$ **do**
      Select random $stag$
      Generate new positions by fighting
      Update $commander$ with best position
    end for
    ▷ Harem Formation
    Calculate normalized fitness for $commanders$
    Divide $hinds$ into harems proportional to fitness
    ▷ Mating Phase
    for all $commander$ in $commanders$ **do**
      Mate with $\alpha\%$ of hinds in own harem
      Mate with $\beta\%$ of hinds from another harem
    end for
    for all $stag$ in $stags$ **do**
      Find nearest hind and mate
    end for
    ▷ Selection Phase
    Update $hinds$ with offsprings via tournament selecion method.
    Update $bestPosition$ and $bestFitness$ if improved
  end for
  return Mapping of cloudlets to VMs using $bestPosition$
end procedure

The algorithm outputs the best mapping of cloudlets to VMs that balances makespan, resource utilization, cost efficiency, and response time.

4.1 Generate initial red deers

Initializing a population of 100 red deer, where each red deer represents a possible solution which is mapping of cloudlet to VMs, refer to figure 3. Each red deer is encoded as an one-dimensional array of size equal to the number of cloudlets, where each element behaves as the index of the VM to be assigned to that particular cloudlet. For instance, a red deer looks like:

$$RD_i = [x_1, x_2, x_3, \ldots, x_{N_c}] \tag{12}$$

Each index in this array corresponds to a cloudlet and the value associated with each index is the VM_id of the VM to whom that cloudlet is being assigned. After the population is generated, the fitness of each red deer is computed using an objective function, which evaluates the performance of the mapping based on makespan, resource utilization, response time, and execution cost.

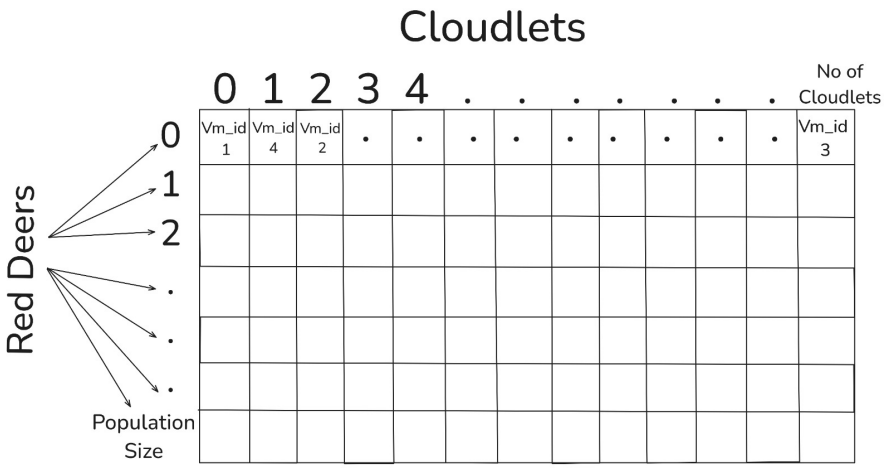

Fig. 3 Mapping of each red deer with one load balancing solution

Once fitness values are obtained, the population is sorted in ascending order of fitness (lower is better), and the top-performing N_{male}(parameter which is already initially set) red deers are selected as males from which top $\gamma\%$ of the males are choosen as commanders, while the remaining form the group of hinds.

Top N_{male} Red Deers are considered as males:

$$M = \{RD_1, RD_2, \ldots, RD_{N_{\text{male}}}\} \tag{13}$$

The remaining individuals are considered as hinds:

$$H = \{RD_{N_{\text{male}}+1}, RD_{N_{\text{male}}+2}, \ldots, RD_{N_{\text{pop}}}\} \tag{14}$$

Where Nmale shows the elitist RDs. Simply said, the number of Nmale maintains the intensification properties, while Nhind considers the diversification phase of the algorithm.

4.2 Roaring Process

In the roaring phase, each male red deer attempts to improve its current solution (i.e., cloudlet-to-VM mapping) by exploring its neighborhood in the solution space. This behavior mimics the natural tendency of red deer to expand their territory and display dominance by roaring.

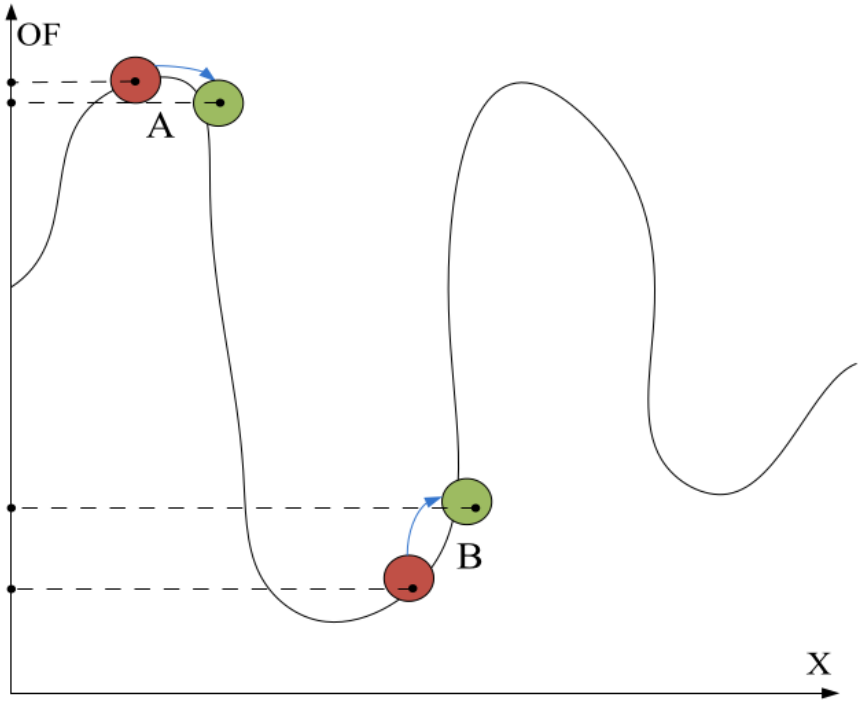

Fig. 4 Roaring Process of male red deers [18]

To simulate this behavior computationally, each male red deer $RD_i \in M$ updates its position according to the following rule [18]:

$$male_i^{\text{new}} = \begin{cases} male_i^{\text{old}} + a_1 \cdot (((UB - LB) \cdot a_2) + LB), & \text{if } a_3 \geq 0.5 \\ male_i^{\text{old}} - a_1 \cdot (((UB - LB) \cdot a_2) + LB), & \text{if } a_3 < 0.5 \end{cases} \tag{15}$$

Where:

- $male_i^{\text{old}}$ is the current red deer position (i.e., current cloudlet-to-VM assignment),
- $male_i^{\text{new}}$ is the updated solution after roaring,

- $UB = N_{VM}$ and $LB = 0$, which denotes the upper and lower bounds of the solution space, respectively. All the numbers in this range represent unique `vm_id`s,
- $a_1, a_2, a_3 \in [0, 1]$ are uniformly distributed random numbers.

Each male RD generates a new position using the above update rule. The updated solution is accepted only if it yields a better fitness value compared to the previous one. This ensures that the exploration during roaring leads to quality improvements in the mapping.

Figure 4 illustrates two scenarios chosen from the solution space to show the effect of roaring in isolation. Observe that cases A and B normally happen within the roaring period. In case A, the new position is accepted because its objective function value (eq.11) is higher than that of the old position. In case B, the new solution is not acceptable. Observe again that the y-axis is the Objective function value, and the x-axis is the positions of males in this figure.

4.3 Selection of the best male RDs as commanders

In nature, not all male red deers possess the same level of strength, dominance, or attractiveness. Some exhibit superior traits and are more successful in claiming and defending territories. This variation is modeled in the RDA by classifying the male red deers into two distinct groups: *commanders* and *stags* [18].

In our cloudlet-to-VM mapping scenario, each male RD represents a potentially optimal mapping configuration. To further intensify the search, we designate the top γ% of male RDs (based on fitness value) as commanders, and the remaining ones as stags.

The number of commanders N_{Com} is computed using the following formula:

$$N_{\text{Com}} = \text{round}(\gamma \cdot N_{\text{male}}) \tag{16}$$

where:

- $\gamma \in [0, 1]$ is a user-defined parameter representing the percentage of elite males to be chosen as commanders,
- N_{male} is the total number of male red deers.

After selecting the commanders, the remaining male RDs are assigned the role of stags. The number of stags N_{stag} is calculated as:

$$N_{\text{stag}} = N_{\text{male}} - N_{\text{Com}} \tag{17}$$

This division of roles creates a hierarchy among the male RDs, which is later used in the fighting and mating phases [18]. In our implementation, we set $\gamma = 0.4$, meaning 40% of male RDs are chosen as commanders which was giving experimentally best outcome.

4.4 Fight between male commanders and stags

Once the male red deers have been divided into commanders and stags, the next step is to simulate the natural behavior of dominance through fights. Each commander is

randomly paired with a stag to engage in a competition to improve the population's overall fitness [18]. These confrontations result in the generation of new candidate solutions through exploitation of the solution space.

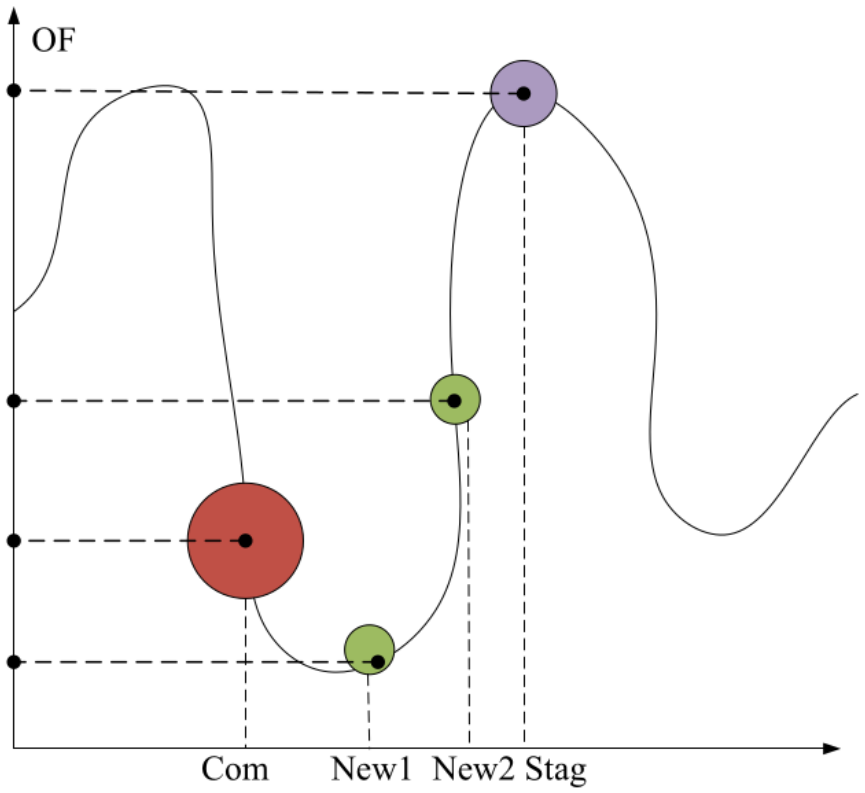

Fig. 5 Fight between a commander and a stag [18]

For each such pairing, two new solutions are generated using the following equations:

$$\text{New}_1 = \frac{(\text{Com} + \text{Stag})}{2} + b_1 \cdot ((\text{UB} - \text{LB}) \cdot b_2 + \text{LB}) \tag{18}$$

$$\text{New}_2 = \frac{(\text{Com} + \text{Stag})}{2} - b_1 \cdot ((\text{UB} - \text{LB}) \cdot b_2 + \text{LB}) \tag{19}$$

where:

- Com and Stag denotes the current position vectors (solutions - Cloudlet to VMs mapping) of the commander and stag, respectively,

- b_1 and b_2 are random variables drawn from a uniform distribution over the interval [0, 1].

The commander is replaced with best solution after evaluating their objective function (eq.11) out of all the four possible solution (commander, stag, and the two new offspring 4.4) [18].

4.5 Form harems

In nature, each male commander claims a group of hinds, forming a harem based on his strength and dominance. In our solution, this behavior is simulated by associating a portion of the hinds population with each commander based on their fitness values. A lower fitness(Lower is better) implies a larger harem.

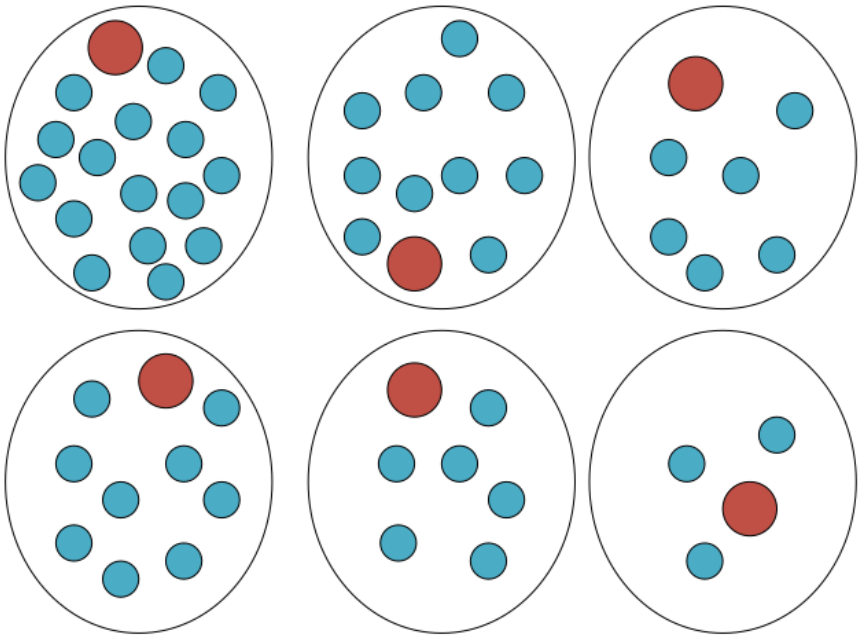

Fig. 6 Harems of each commander [18]

Calculate the relative power V_n of each commander using the following equation:

$$V_n = v_n - \max\{v_i\} \tag{20}$$

where:

- v_n is the fitness value (objective function value) of the n-th commander,
- $\max\{v_i\}$ represents the worst (here, maximum) fitness among all commanders, ensuring normalization by fitness proximity to optimality.

Then, we normalize this value to determine the portion of hinds to be assigned to each commander:

$$P_n = \frac{V_n}{\sum_{i=1}^{N_{Com}} V_i} \tag{21}$$

where:

- P_n is the normalized power of the n-th commander,
- N_{Com} denotes the total number of commanders.

Now, based on this proportional power, the number of hinds allocated to the n-th commander's harem is computed as:

$$N.\text{harem}_n = \text{round}(P_n \cdot N_{\text{hind}}) \tag{22}$$

where:

- $N.\text{harem}_n$ is the number of hinds of the n-th commander's harem,
- N_{hind} is the total number of hinds in the population.

Hinds are then randomly assigned to commanders based on the computed values $N.\text{harem}_n$. Each selected group of hinds, together with its corresponding commander, constitutes one harem.

4.6 Mating of commanders with hinds in his harem

Like other species in nature, red deer mates with each other. Here commander mates with α% of hinds of his harem and generates offsprings. It is calculated as:

$$N.\text{Harem}_n^{mate} = \text{round}(\alpha \cdot N.\text{Harem}_n) \tag{23}$$

where:

- $N.\text{Harem}_n^{mate}$ is the number of hinds from the n-th harem that will mate with the commander,
- $N.\text{Harem}_n$ is the total number of hinds in that harem,
- α [0-1] is a user-defined uniform model parameter that controls the mating percentage.

Once the hinds for mating are selected randomly from each harem, a new offspring (solution) is produced using the following equation:

$$\text{offs} = \frac{\text{Com} + \text{Hind}}{2} + c \cdot (UB - LB) \tag{24}$$

where:

- Com and Hind represent the current position vectors (solutions - Cloudlet to VMs mapping) of the commander and a selected hind,
- offs is the new offspring (i.e., the new candidate solution),
- c is a random number generated from a uniform distribution over the interval [0, 1].

4.7 Mating of commander with hinds of another harem

Allow each commander to mate with $\beta\%$ of hinds from another randomly selected harem [18].

Let i be the index of a randomly selected harem which is not itself. The number of hinds in the i-th harem that mate with the current commander is given by:

$$N.\text{Harem}_i^{mate} = \text{round}(\beta \cdot N.\text{Harem}_i) \tag{25}$$

where:

- $N.\text{Harem}_i^{mate}$ is the number of hinds from the i-th harem that will mate with the commander,
- $N.\text{Harem}_i$ is the number of hinds in the i-th harem,
- β is a user-defined parameter controlling the proportion of hinds involved in inter-harem mating, range value of this parameter is between zero and one.

Each of the $N.\text{Harem}_i^{mate}$ hinds selected from the i-th harem mates with the current commander using the same offspring generation formula defined in 4.6.

4.8 Mating of stag with the nearest hind

Unlike commanders, stags do not form harems but still participate in the mating phase. Each stag mates with the hind who is nearest.

The nearest hind is determined using Euclidean distance in the J-dimensional solution space. The distance between a stag and the i-th hind is calculated as:

$$d_i = \sqrt{\sum_{j=1}^{n} (\text{stag}_j - \text{hind}_{i,j})^2} \tag{26}$$

where:

- d_i is the distance between the stag and the i-th hind,
- stag_j and $\text{hind}_{i,j}$ represent the j-th dimension of the stag and of the i-th hind, respectively,

After identifying the nearest hind (i.e., the one with the smallest d_i), the stag mates with it using the same equation as 4.6, replacing the commander's role with that of the stag.

4.9 Select the next generation

The next generation is formed by combining the current solutions and newly generated offsprings.

To achieve this, two selection strategies are used:

1. **Elite Preservation:** All male red deer from the current generation, including both commanders and stags, are retained. These represent the elite solutions.
2. **Offspring Selection:** The remainder of the population is filled by selecting individuals from the pool of all hinds and the offspring generated through the mating processes. This selection is performed using tournament selection method.

4.10 Stopping condition

RDA continues its iterative process till 100 iterations which is a optimal value experimentally proven[18].

Figure 1 and Alogrithm 2 provides a summarized overview of the main steps involved in the RDA. It highlights the initialization, male classification, roaring, fighting, harem formation, mating, and selection processes that collectively guide the search for optimal load balancing in the cloud environment.

5 Experimental Setup

Evaluating the performance of the implemented resource provisioning algorithms, including our proposed approach, in a consistent and repeatable manner across different system environments and configurations was a challenging task. To address this, we used CloudSim to simulate a virtual cloud environment. CloudSim is an open and extensible simulation framework via which cloud computing infrastructures and resource provisioning policies can be simulated and modelled. It was created at the CLOUDS Laboratory, Department of Computer and Software Engineering, University of Melbourne [30], it provides a controlled environment to conduct experiments without the need for a real cloud infrastructure. It allowed us to compare the performance of our proposed algorithm with other existing algorithms effectively. Moreover, CloudSim supports customization by enabling developers to extend or replace its core classes, making it suitable for implementing new scheduling or provisioning policies [31].

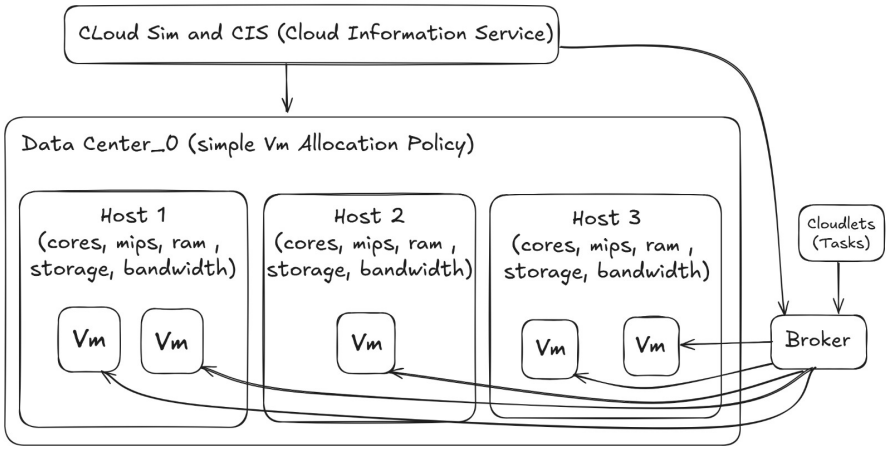

Fig. 7 Basic architecture of cloudSim

5.1 Simulation data and parameters

To simulate a realistic cloud environment, we configured three hosts with varying hardware specifications, as shown in Table 2. These configurations were chosen to

Table 2 Technical Characteristics of the hosts

Host ID	Cores	Processing speed, MIPS	RAM, MB 4	Storage, MB	Bandwidth, Mbps
1	4	5000	204,800	1,048,576	102,400
2	2	2500	102,400	1,048,576	102,400
3	1	1000	51,200	1,048,576	102,400

Table 3 Parameters of RDA

Parameters	Value
Population size(no. of solutions)	100
Male population(no. of males)	15
Maximum iteration	100
Alpha(% of hinds a commander mates with in his harem)	0.5
Beta(% of hinds a commander mates with in another harem)	0.2
Gamma(Fraction of males selected as commanders)	0.4
LB(Lower Bound)	0.0
UB(Upper Bound)	N_{VM}

represent heterogeneous cloud resources, which is a common scenario in real-world data centers. Each host varies in terms of the number of CPU cores, processing speed (MIPS), RAM, and other attributes, allowing us to test the algorithm's performance under diverse conditions.

For RDA, the parameters listed in Table 3 were finalized after multiple rounds of experimentation. Various combinations of population size, male-to-female ratio, and mating behavior percentages (α, β, γ) were tested. The configuration yielding the most stable and optimal results in terms of makespan, response time, and resource utilization was selected. Specifically, a population size of 100 and 15 males with a commander selection ratio of 40% provided a good balance between exploration and exploitation. The bounds for solution values were defined based on task and resource limits in the simulation environment.

All the experiments were performed on a Lenevo yoga 6 laptop with intel i5 13^{th} generation H series processor and 16 gigabytes Ram.

6 Results and Analysis

The RDA algorithm is especially good at reducing makespan by a synergy of smart design elements competing against one another. Combining RDA with QoS, it performs well in discovering new possibilities while optimizing current solutions. Its nature-inspired phases—roaring, fighting, and mating—stimulate diverse ideas without leaping too quickly at suboptimal solutions. The algorithm organizes its population hierarchically, separating males into commanders and stags, and hinds into harems based on their performance. This organization has the highest-performing members contributing more to future generations, leading the group towards stronger solutions.

Furthermore, selective replacement keeps only the best offspring, always improving solution quality. Overall, RDA's combination of targeted fitness checks, organized population dynamics, and selective selection makes it very effective in reducing makespan. The GWO is also successful in reducing makespan, with a leadership-based search strategy where the best solutions lead the population to promising regions. Its hierarchical organization is similar to that of a wolf pack, with alpha, beta, and delta wolves guiding the members of the sub-order towards quick convergence. GWO maintains exploration-exploitation balance through adaptive parameter adjustment at no computational cost of sophisticated genetic operators like crossover or mutation. The GWO is more probable to generate superior solution quality compared to GA since its reduced search strategy is robust while converging to high-performance configurations with improved accuracy. This is demonstrated more clearly in Figure 8.

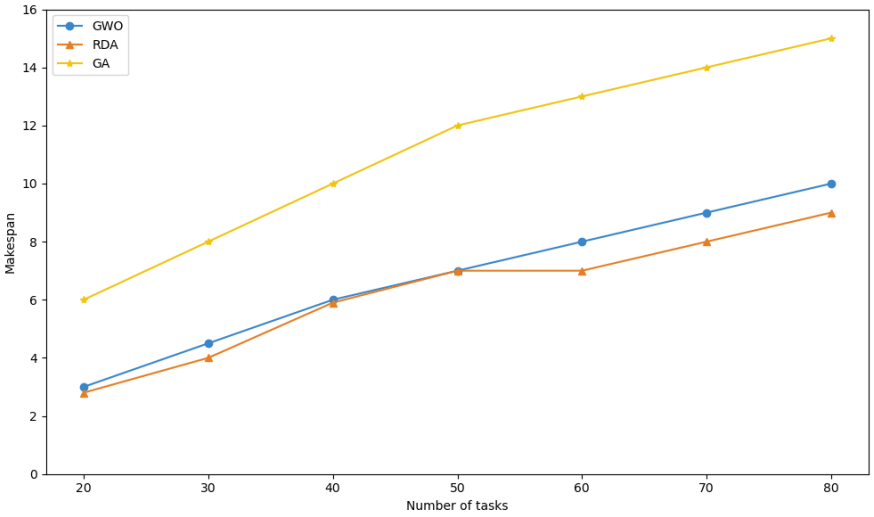

Fig. 8 Makespan Comparison

Figure 9 presents a detailed comparison of response times across three algorithms under evaluation, with task counts configured at 20, 30, 40, 50, 60, 70, and 80. Response time, a critical metric to minimize, is calculated by aggregating the execution durations of individual services to determine the total time required for composite function processing. The results, as depicted in the figure, underscore the superior efficiency of the proposed algorithm in optimizing response time.

To evaluate the convergence behavior of the RDA, it was pitted against GA and GWO over 100 iterations. As illustrated in Figure 10, the convergence test involved 80 tasks, each with 200 potential services, across 100 iterations, highlighting how the algorithms stack up in terms of solution refinement.

The RDA offers a highly effective approach to load balancing in distributed, grid, and cloud computing environments, offering distinct advantages over traditional approaches through its optimization framework. By modeling the complex mating

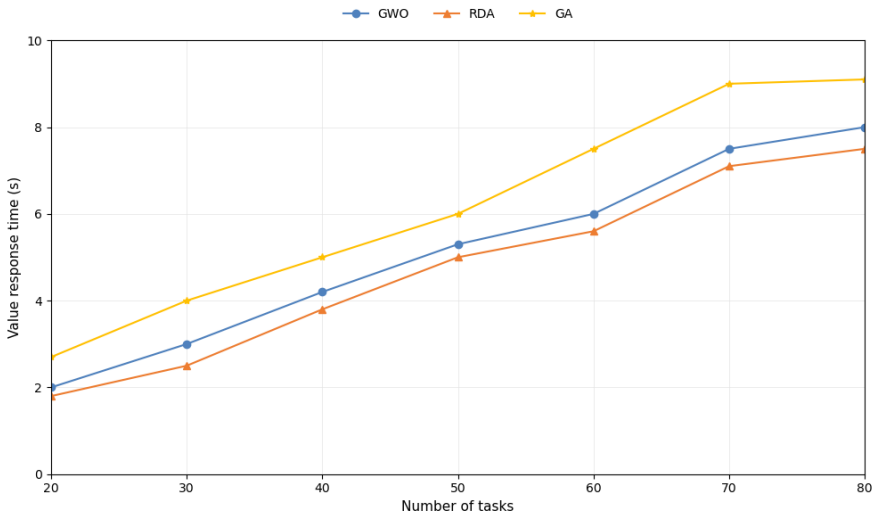

Fig. 9 Response Time Comparison

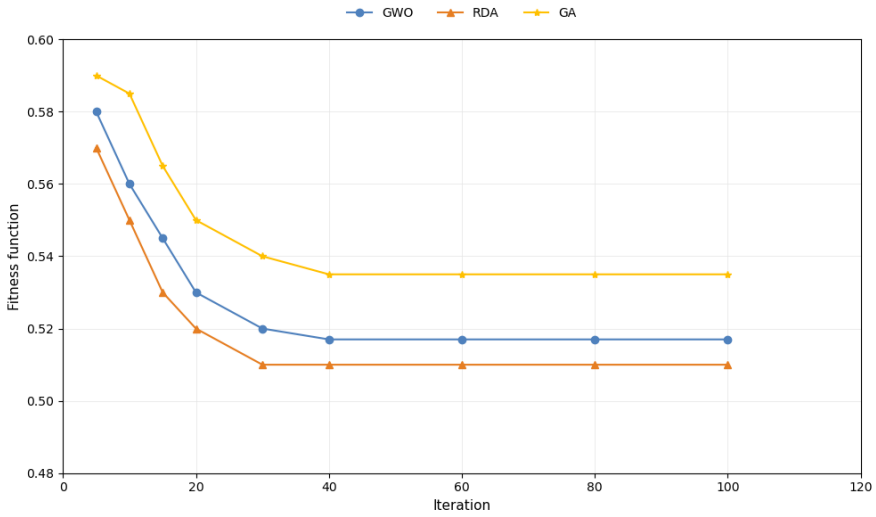

Fig. 10 Convergence Comparison

rituals of red deer—particularly their roaring competitions, territorial conflicts, and harem establishment processes—RDA effectively navigates the crucial balance between exploration and exploitation phases that often challenge other algorithms. When used in cloud resource allocation, RDA is very effective in the way that it considers both existing system loads and QoS demands simultaneously, thus avoiding resource saturation and underutilization. Our work confirms that RDA consistently outperforms rival evolutionary algorithms such as the GWO and GA, particularly at high loads. The

strongest aspect of the algorithm is its ability to dynamically adapt based on evolving conditions. Our simulation evidence in real data attests to the improved performance of RDA on vital parameters such as reduced makespan and enhanced response time, although rival algorithms occasionally display increased fitness values in specific constraint cases. The combination of quality-driven resource allocation with principles of biologically-inspired optimization allows RDA to solve the complicated issues inherent in current cloud computing. The integration offers a solution that, apart from optimizing task execution efficiency, balances the system irrespective of the loading status. Such results indicate that RDA offers an incredibly practical way of solving computing systems for which timely provision of services and balancing workloads are critical operational needs.

7 Conclusion and Future Work

This study evaluates the RDA for cloud load balancing, demonstrating that this nature-inspired method effectively addresses resource allocation challenges and outperforms conventional and other metaheuristic algorithms like GWO and GA in reducing makespan time, and response times and in increasing cost efficiency, and resource utilization primarily due to its balanced exploration and exploitation capabilities; furthermore the algorithm presents avenues for enhancement through integration with machine learning and multi-criteria decision making, along with evaluations of expanded systems and additional QoS factors to bolster real-world cloud adaptability. In the future, studies can explore whether RDA can be used together with other smart algorithms or methods. It can also examine its performance on extremely large data sets and compare its performance with deep learning algorithms to determine how they can complement each other. Future studies can also explore how the fitness function can be enhanced by incorporating other elements such as Jain's fairness Index [32] and exploring how stable the algorithm's reliability is and how stable the quality of the service is.

References

[1] Mell, P., Grance, T.: The nist definition of cloud computing. Technical Report 800-145, National Institute of Standards and Technology, Gaithersburg, MD (September 2011). https://doi.org/10.6028/NIST.SP.800-145 . https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf

[2] Kumar, J., Rani, A., Dhurandher, S.: Convergence of user and service provider perspectives in mobile cloud computing environment: Taxonomy and challenges. International Journal of Communication Systems **33** (2020) https://doi.org/10.1002/dac.4636

[3] Ahmad, I., Bakht, H., Mohan, U.: Cloud computing – a comprehensive definition. Journal of Computing and Management Studies **1**(1), 1–6 (2017)

[4] Kumar, J., Singh, A.: Performance evaluation of metaheuristics algorithms for workload prediction in cloud environment. Applied Soft Computing **113**, 107895 (2021) https://doi.org/10.1016/j.asoc.2021.107895

[5] Shahid, M.A., Islam, N., Alam, M.M., Su'ud, M.M., Musa, S.: A comprehensive study of load balancing approaches in the cloud computing environment and a novel fault tolerance approach. IEEE Access **8**, 130500–130526 (2020) https://doi.org/10.1109/ACCESS.2020.3009184

[6] Alicherry, M., Lakshman, T.V.: Optimizing data access latencies in cloud systems by intelligent virtual machine placement. IEEE Transactions on Parallel and Distributed Systems **25**(3), 647–655 (2013) https://doi.org/10.1109/INFCOM.2013.6566850

[7] Boulmier, A., Abdennadher, N., Chopard, B.: Optimal load balancing and assessment of existing load balancing criteria. Journal of Parallel and Distributed Computing **169**, 211–225 (2022) https://doi.org/10.1016/j.jpdc.2022.07.002

[8] Attallah, S.M.A., Fayek, M.B., Nassar, S.M., Hemayed, E.E.: Proactive load balancing fault tolerance algorithm in cloud computing. Concurrency and Computation: Practice and Experience **33**(10), 6172 (2021) https://doi.org/10.1002/cpe.6172

[9] Mohammadi Golchi, M., Saraeian, S., Heydari, M.: A hybrid of firefly and improved particle swarm optimization algorithms for load balancing in cloud environments: Performance evaluation. Computer Networks **162**, 106860 (2019) https://doi.org/10.1016/j.comnet.2019.106860

[10] Zhou, J., Kumar Lilhore, D., Tao, H., Simaiya, S., Jawawi, D., Alsekait, D., Ahuja, S., Biamba, C., Hamdi, M.: Comparative analysis of metaheuristic load balancing algorithms for efficient load balancing in cloud computing. Journal of Cloud Computing **12** (2023) https://doi.org/10.1186/s13677-023-00453-3

[11] Blum, C., Roli, A.: Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. Springer, ??? (2003)

[12] Mansouri, N., Zade, B.M.H., Javidi, M.M.: Hybrid task scheduling strategy for cloud computing by modified particle swarm optimization and fuzzy theory. Computers & Industrial Engineering **130**, 597–633 (2019) https://doi.org/10.1016/j.cie.2019.03.006

[13] Li, K., Zomaya, A.Y.: Energy-aware scheduling for cloud computing: A survey. ACM Computing Surveys (CSUR) **51**(3), 1–35 (2018) https://doi.org/10.1145/3186832

[14] Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of ICNN'95 - International Conference on Neural Networks, pp. 1942–19484 (1995).

https://doi.org/10.1109/ICNN.1995.488968

[15] Dorigo, M.: Optimization, learning and natural algorithms (in italian). PhD thesis, Politecnico di Milano, Italy (1992)

[16] Yang, X.-S.: A brief introduction to nature-inspired metaheuristic algorithms. Studies in Computational Intelligence **284**, 1–21 (2010) https://doi.org/10.1007/978-3-642-04944-6_1

[17] Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems **25**(6), 599–616 (2009) https://doi.org/10.1016/j.future.2008.12.001

[18] Fathollahi-Fard, A.M., Hajiaghaei-Keshteli, M., Tavakkoli-Moghaddam, R.: Red deer algorithm (rda): a new nature-inspired meta-heuristic. Soft computing **24**, 14637–14665 (2020)

[19] Sefati, S., Navimipour, N.J.: A qos-aware service composition mechanism in the internet of things using a hidden-markov-model-based optimization algorithm. IEEE Internet of Things Journal **8**(20), 15620–15627 (2021) https://doi.org/10.1109/JIOT.2021.3074499

[20] Pradhan, B., Das, D., Sahoo, S.: An efficient red deer algorithm for multi-objective optimization problems. Journal of Ambient Intelligence and Humanized Computing, 1–17 (2022) https://doi.org/10.1007/s12652-022-03714-w

[21] Kruekaew, B., Kimpan, W.: Enhancing of artificial bee colony algorithm for virtual machine scheduling and load balancing problem in cloud computing. International Journal of Computational Intelligence Systems **13** (2020) https://doi.org/10.2991/ijcis.d.200410.002

[22] KOKILAVANI, T.: Load balanced minmin algorithm for static metatask scheduling in grid computing. International Journal of Computer Applications **20** (2011) https://doi.org/10.5120/2403-3197

[23] Goyal, S., Singh, M.: Adaptive and dynamic load balancing in grid using ant colony optimization. International Journal of Engineering and Technology **4** (2012)

[24] Makasarwala, H., Hazari, P.: Using genetic algorithm for load balancing in cloud computing. IEEE Access **4**, 7700–7705 (2016) https://doi.org/10.1109/ECAI.2016.7861166

[25] Sefati, S.S., Mousavinasab, M., Farkhady, R.: Load balancing in cloud computing environment using the grey wolf optimization algorithm based on the reliability: performance evaluation. The Journal of Supercomputing **78** (2022) https://doi.

org/10.1007/s11227-021-03810-8

[26] Devaraj, A., Elhoseny, M., Dhanasekaran, S., Lydia, E., Shankar, K.: Hybridization of firefly and improved multi-objective particle swarm optimization algorithm for energy efficient load balancing in cloud computing environments. Journal of Parallel and Distributed Computing **142**, 36–45 (2020)

[27] Lilhore, U., Simaiya, S., Maheshwari, S., Manhar, A., Kumar, S.: Cloud performance evaluation: hybrid load balancing model based on modified particle swarm optimization and improved metaheuristic firefly algorithms (2020)

[28] Shi, L., Wang, X., Ma, R.T., Tay, Y.C.: Weighted fair caching: Occupancy-centric allocation for space-shared resources. ACM SIGMETRICS Performance Evaluation Review **46**(3), 35–36 (2019)

[29] Zanbouri, K., Navimipour, N.: A cloud service composition method using a trust-based clustering algorithm and honeybee mating optimization algorithm. International Journal of Communication Systems **33** (2019) https://doi.org/10.1002/dac.4259

[30] Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A., Buyya, R.: 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 (2011)

[31] Goyal, T., Singh, A., Agrawal, A.: Cloudsim: simulator for cloud computing infrastructure and modeling. Procedia Engineering **38**, 3566–3572 (2012) https://doi.org/10.1016/j.proeng.2012.06.412 . INTERNATIONAL CONFERENCE ON MODELLING OPTIMIZATION AND COMPUTING

[32] Rezaeinia, N., Góez, J.C., Guajardo, M.: On efficiency and the jain's fairness index in integer assignment problems. Computational Management Science **20**(1), 42 (2023)