Chapter 4

Multiobjective Task Scheduling Algorithms

4.1 Introduction

In this chapter, we delve into the realm of Multiobjective Task Scheduling Algorithms, specifically focusing on the utilization of the Grey Wolf Optimization (GWO) and Whale Optimization Algorithm (WOA). Multiobjective optimization involves the simultaneous consideration of multiple conflicting objectives, which is a common challenge in cloud computing task scheduling. The GWO and WOA algorithms, known for their inspiration drawn from the behavior of animals, are introduced as key components of the multiobjective task scheduling framework.

This chapter explores the intricacies of multiobjective optimization, emphasizing the importance of achieving a balance between conflicting objectives such as cost minimization and task completion time. GWO and WOA, two powerful optimization techniques, are employed to address this complex problem in cloud environment. Their unique methodologies, inspired by the collaborative and exploratory behavior of wolves and whales, contribute to the development of efficient and effective multiobjective task scheduling algorithms.

Through a series of simulations, comparisons, and analyses, this chapter aims to showcase the prowess of GWO and WOA in tackling the challenges posed by multiobjective task scheduling. The subsequent sections unfold the algorithmic details, experimental setups, and comparative results, providing insights into the novel approach of leveraging nature-inspired algorithms for optimizing cloud computing task scheduling with multiple conflicting objectives.

4.2 Multi-objective User Defined Weight Based Scheduling Using Grey Wolf Optimisation In Cloud

This section presents a new method focusing on the Grey Wolf Optimization (GWO) algorithm for workflow scheduling in a cloud environment. Our approach works in a multi-objective optimization framework, aiming at optimizing operational cost, time, and load balancing. The main objective of our proposed method is to enable users to fine-tune their workflow design priorities, evaluating the effectiveness of our method to assign different weights for the optimization of various objectives.
The results of our experiments show the remarkable effectiveness of our approach in reducing the overall cost and implementation time along with load balancing. This chapter contributes to the advancement of cloud computing by matching versatile and efficient workflow scheduling approach to users' specific needs and priorities, ultimately improving the use of cloud computing resources.

4.2.1 Problem Statement

The main objective is to propose a new multi-objective workflow scheduling strategy in the cloud based on Grey Wolf optimization. The algorithm aims to optimize the performance by simultaneously considering cost, makespan, and load balancing. While minimizing time and cost are indeed crucial objectives for end users, incorporating load balancing ensures the overall efficiency and stability of the system. This ensures optimal resource utilization and prevents any single resource from being overwhelmed, ultimately enhancing the system's performance and reliability.

Mathematical Formulation

Decision Variables: Let's denote:

- T: Set of tasks, where $T = T_1, T_2, \ldots, T_n$
- M: Set of virtual machines (VMs), where $M = M_1, M_2, \ldots, M_k$

Constraints:

i **Task Assignment Constraint:**

- Each task should be assigned to exactly one virtual machine.
- Sum of assignment variables for each task T_i should be equal to 1: $\Sigma\, x_{t_i} = 1$, $\forall t_i \in$ T

ii **Load Balancing Constraint:**

- The load on each VM should be balanced based on the load balancing weight.
- The load on each VM M_j should be less than or equal to the average load multiplied by the load balancing weight: $\Sigma\ (x_{t_i} * L_{t_i}) \leq y_m * (1/k) * \Sigma(L_{t_i}), \forall M_j \in M$

iii **Resource Constraint:**

- The resource requirements of assigned tasks should not exceed the capacity of the VM: $\Sigma(x_{t_i} * R_{t_i}) \leq C_m, \forall M_j \in M$

Objective Functions:

- Minimize Cost: Minimize the total cost associated with task scheduling in the cloud.
- Minimize Time: Minimize the makespan or total execution time of all tasks in the cloud.
- Load Balancing: Optimize the load balancing across virtual machines.

$$\text{Minimize } f(x) = W_1 \cdot C(x) + W_2 \cdot T(x) + W_3 \cdot L(x) \tag{4.1}$$

Subject to:

$$\Sigma x_{t_i} = 1, \forall T_i \in T \tag{4.2}$$

$$\Sigma(x_{t_i} * L_{t_i}) \leq y_m * (1/k) * \Sigma(L_{t_i}), \forall M_j \in M \tag{4.3}$$

$$\Sigma(x_{t_i} * R_{t_i}) \leq C_m, \forall M_j \in M \tag{4.4}$$

$$x_{t_i}, y_m \geq 0, \forall T_i \in T and M_j \in M \tag{4.5}$$

In this formulation, $C(x)$ represents the total cost associated with task scheduling, $T(x)$ represents the makespan or total execution time of tasks, $L(x)$ represents the load balancing measure, and W_1, W_2, and W_3 are the weighting factors for each objective. The fitness function $f(x)$ combines these objectives to be minimized.
The mathematical equations for calculating $C(x)$, $T(x)$, and $L(x)$, where x represents the schedule as explained in Fig. 1.4.

- Let's denote the cost of task T_i on VM M_j as $Cost(t_i, m_j)$. Then, the total cost $C(x)$ can be calculated as:

$$C(x) = \Sigma(Cost(T_i, M_j) * X_{ij}), \forall T_i, M_j \tag{4.6}$$

- Let's denote the execution time of task T_i on VM M_j as $ExecTime(T_i, M_j)$. Then, the makespan $T(x)$ can be calculated as:

$$T(x) = \max\left(\sum_i \sum_j \left(\text{ExecTime}(T_i, M_j) \cdot X_{ij}\right), \forall T_i, M_j\right) \tag{4.7}$$

- Let's denote the load of VM M_j as $Load(M_j)$. Then, the load balancing measure $L(x)$ can be calculated as:

$$L(x) = \max\left(\text{Load}(M_j)\right) - \frac{1}{k}\sum \text{Load}(M_j), \forall M_j \tag{4.8}$$

  we calculate the Load of each VM by:

$$Load(M_j) = \frac{N * cloudletLength}{VM_{mips}} \tag{4.9}$$

  where j is the j^{th} VM, N is the number of task allocated to the VM, $cloudletLength$ is the length of tasks specified in million instructions (MI) and VM_{mips} is MIPS rate of VM.

In these equations, X_{ij} is a binary decision variable that represents whether task T_i is assigned to VM M_j. $X_{ij} = 1$ if task T_i is assigned to VM M_j, and $X_{ij=0}$ otherwise. T_i and M_j represent tasks and VMs, respectively. $Cost(T_i, M_j)$ represents the cost of assigning task T_i to VM M_j, and $ExecTime(T_i, M_j)$ represents the execution time of task T_i on VM M_j. $Load(M_j)$ represents the load on VM M_j, and k represents the total number of VMs.

4.2.2 Proposed Methodology

Scheduling workflows in the cloud can be a challenging optimisation challenge that requires balancing a number of competing goals, including cost, time, and load. The Grey Wolf Optimisation (GWO) algorithm, a population-based metaheuristic optimisation algorithm inspired by the hunting behaviour of grey wolves, is one potential solution to this challenge. The fundamental concept behind GWO is to mimic the grey wolf's hunting style in order to solve an optimisation problem in the best possible way. An initial population of potential solutions—referred to as "wolves"—is initialised by the algorithm. The location of each wolf in the population, which represents a potential answer to the issue, is determined by a set of decision variables. The wolves communicate with one another and adjust their postures in accordance with a set of predetermined rules during the optimisation process. These regulations are intended to resemble how wolves interact with one another, with the alpha wolf serving as the pack's leader and the beta and delta wolves serving as its followers. We need to create an objective function that captures the trade-off between cost, time, and load in order to apply GWO to workflow scheduling in the cloud. The following

formulation fits this objective function:

$$\text{Minimize } f(x) = W_1 \cdot C(x) + W_2 \cdot T(x) + W_3 \cdot L(x) \tag{4.10}$$

where $C(x)$ is the cost of executing the workflow using the scheduling solution x, $T(x)$ is the time required to complete the workflow using the scheduling solution x, and $L(x)$ is the load balancing factor of the scheduling solution x. The weights W_1, W_2, and W_3 are tuning parameters that control the relative importance of each objective.

The steps below can be used to use GWO to address the optimisation problem:

1. Randomly initialise a wolf population in the search area.

2. Employ the objective function to assess each wolf's fitness.

3. Adjust the $\alpha, \beta, and \delta$ wolves' locations in accordance with their fitness scores.

4. Use the equation below to update the positions of the remaining wolves:

$$x_i(t+1) = (x_\alpha(t) + A_1 \cdot D_\alpha \cdot r_1) + (x_\beta(t) + A_2 \cdot D_\beta \cdot r_2) + (x_\delta(t) + A_3 \cdot D_\delta \cdot r_3) \tag{4.11}$$

   where $x_i(t+1)$ is the position of the i^{th} wolf at time $t+1$, $x_\alpha(t)$, $x_\beta(t)$, and x_δ are the positions of the alpha, beta, and delta wolves at time $t, A_1, A_2,$ and A_3 are coefficient vectors, D_α, D_β, and D_δ are the distances between the i^{th} wolf and the alpha, beta, and delta wolves, and r_1, r_2, and r_3 are random vectors.

5. Apply the objective function to assess each updated wolf's fitness as explained in Eq. 4.10.

6. Continue with steps 3-5 again till a stopping point is reached.

The outcome is a scheduling option that strikes a balance between the goals of cost, time, and load. Depending on the demands of the application, we can prioritise one target above the others by modifying the weights W_1, W_2, and W_3.

4.2.3 Implementation and Results

This section shows the simulation parameters and results obtained for different objectives and ends with the comparsion of cost, time and load weights variations from 0 to 100.

Simulation Parameters

CloudSim [16] is used to evaluate results. The paramters which were used for the evaluation has been given in Table 4.1, which involve all the configurations for simulation.

Algorithm 3 Workflow Scheduling using Grey Wolf Optimization

Create a directed acyclic graph (DAG) representation of the workflow ▷ Nodes are tasks, edges are dependencies
Initialize population of candidate solutions using GWO method
while stopping criteria not met **do**
    for each candidate solution in population **do**
        Determine cost, time, and load balancing values
    Arrange population according to objective function as explained in Equation 4.1
    for each candidate solution in population **do**
        Produce new candidate solutions using Grey Wolf Optimization algorithm(Algorithm 4)
        Assess fitness of new candidates
        Incorporate new candidates into population
        Eliminate worst-performing solutions to preserve population size
Choose the best solution from the final population as the optimal scheduling solution

Algorithm 4 Grey Wolf Optimization Algorithm

Initialize wolf population randomly within the search area
for each wolf in population **do**
    Calculate fitness of the wolf using the objective function
while stopping condition is not met **do**
    Find $\alpha, \beta, and \delta$ wolves with highest fitness
    for each wolf in population (excluding α, β, δ) **do**
        Generate random vectors $r1$, $r2$, and $r3$
        Calculate new position using equation 4 and coefficients $A1$, $A2$, $A3$
        Update the position of the wolf
    for each updated wolf in population **do**
        Calculate fitness using the objective function
Identify wolf with highest fitness as the optimal solution

Parameter	Value
Scheduling algorithms	FCFS, Jaya, Harmony search, PSO, GA, GWO algorithm
Number of Virtual Machines	3,5,10,15,20
Count of datacenter	1
Cost for Tranmission/byte	0.01$
Cost for Processing	3.00$
RAM for VM	128, 512 and 1024 MB
Size of file	100-300 MB
OS for Data Center	LINUX
No. of cloudlets	10,50,100,500,1000
No. of CPU	1
Bandwidth	1000-2000 MB

Table 4.1 CloudSim parameters

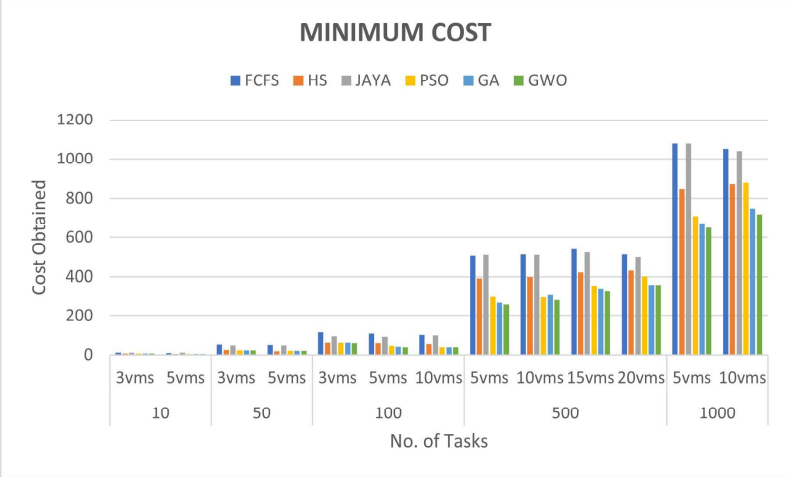

Fig. 4.1 Minimum cost obtained after varying tasks and VMs

Cost Based GWO Algorithm

In the results, GWO algorithm has the best performance while varying tasks and VMs. FCFS algorithm has the worst performance. Jaya and Harmony Search have good results but most comparable results woth GWO are of GA and PSO. Table 4.2 shows the exact values obtained and the Fig. 4.1 shows the comparison in graphical format.

Makespan Based GWO Algorithm

For almost every schedule, GWO algorithm has the shortest execution time. GA and PSO results are comparable with GWO results but FCFS performs the worst. Jaya and HS performs much worse than GWO. Table 4.3 displays the average makespan values obtained

MINIMUM COST							
no of Tasks	**no of Vms**	**FCFS**	**HS**	**JAYA**	**PSO**	**GA**	**GWO**
10	3vms	10.53	5.82	10.38	5.85	5.85	**5.85**
	5vms	9.04	4.85	10.61	4.03	4.03	**4.03**
50	3vms	52.73	23.45	49.63	22.47	22.49	**22.45**
	5vms	51.17	18.43	49.2	19.89	20.15	**19.88**
100	3vms	114.04	62.36	92.99	62.1	62.24	**60.62**
	5vms	107.78	59.99	91.97	47.95	42.99	**41.06**
	10vms	101.22	55.27	97.92	40.42	40.32	**40.08**
500	5vms	507.88	391.52	511.52	298.08	268.96	**259.78**
	10vms	514.75	399.04	512.03	294.85	307.06	**282.27**
	15vms	542.1	420.02	526.06	351.94	334.75	**324.47**
	20vms	513.7	429.95	500.05	400.06	360.04	**359.78**
1000	5vms	1078.96	845.15	1078.81	708.12	672.72	**649.78**
	10vms	1050.73	870.47	1040.78	878.12	747.1	**718.07**

Table 4.2 Cost comparisons for various algorithms

and the Figure 4.2 shows the comparison in graphical format.

Load Based GWO Algorithm

The results of the load balancing demonstrate that GWO algorithm is the most effective for nearly all schedules. GA algorithm performs similarly to GWO, while FCFS algorithm is the least efficient. PSO, Jaya and HS algorithms are not as proficient as GWO. Table 4.4 displays the mean load values obtained and the Fig. 4.3 shows the comparison in graphical format.

The results show that the GWO algorithm performs better as compared to other in terms of Cost and Makespan. The results show that GWO is almost 10-20% better than PSO and almost 5-10% better than GA which can be clearly seen in the tables above. The results show that GWO works better for task scheduling than HS and Jaya without any pre processing by almost 18-25%. FCFS performs worst which is 40-50% less efficient than GWO.

The below Table 4.5 shows the variations of user defined weights and values of cost, time and average load %. The trade-off can be clearly seen and user can use their values of weight according to their requirements.

The Fig. 4.4 shows the normalised values of various objectives for better comparison. It can be seen clearly that the variation among weights is making any of the objective change accordingly and the other objectives are still performing comparable. The formula

MINIMUM MAKESPAN							
no of Tasks	**no of Vms**	**FCFS**	**HS**	**JAYA**	**PSO**	**GA**	**GWO**
10	3	16.94314	9.942833	11.73134	9.17426	10.04422	**10.00101**
	5	27.68262	7.607174	9.980169	8.043929	8.715401	**7.718459**
50	3	126.6769	43.88912	50.85849	42.77712	44.82464	**42.79179**
	5	126.526	36.15251	45.38362	32.94607	33.46841	**34.33514**
100	3	242.1613	83.58032	112.2842	94.08911	82.02565	**84.29377**
	5	263.9227	69.84558	75.92232	70.57329	66.94732	**66.47304**
	10	147.7672	40.87016	52.91693	38.57321	36.88685	**36.08385**
500	5	1239.696	412.0067	801.7674	334.4381	329.6201	**329.0625**
	10	743.4034	259.4248	646.5148	199.4934	181.664	**155.9656**
	15	520.0604	241.2253	529.9172	140.0051	136.5635	**122.1653**
	20	403.3105	175.0504	401.5091	112.1205	108.3061	**110.2708**
1000	5	2449.573	1045.45	2006.885	802.0894	661.1509	**602.8079**
	10	1492.656	644.1991	1579.586	501.4768	364.6147	**362.2978**

Table 4.3 Makespan comparisons for various algorithms

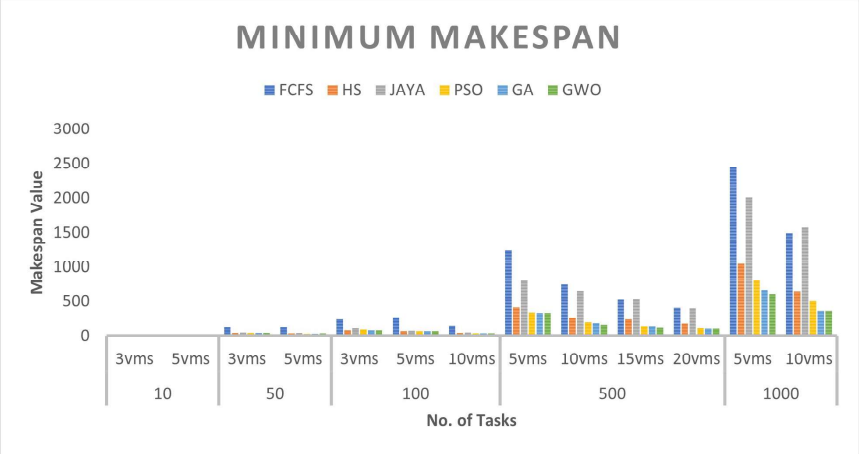

Fig. 4.2 Minimum makespan obtained after varying tasks and VMs

AVERAGE LOAD							
no of Tasks	**no of Vms**	**FCFS**	**HS**	**JAYA**	**PSO**	**GA**	**GWO**
10	3	5.76129	5.19098	6.83543	5.7654	5.773	**3.37157**
	5	4.80779	4.1112	3.8829	3.9289	2.2668	**2.81076**
50	3	35.7515	23.3605	27.1634	28.934	26.826	**18.7674**
	5	23.6071	22.8233	16.3786	18.393	12.716	**11.4405**
100	3	62.1014	46.7414	69.0426	72.998	50.647	**53.5129**
	5	45.2583	34.701	43.7958	25.207	36.652	**22.5631**
	10	19.4782	20.0066	17.6638	18.37	11.663	**12.2902**
500	5	196.99	151.168	221.815	121.99	113.65	**133.717**
	10	96.8092	91.9334	83.4775	72.17	95.759	**57.322**
	15	68.0885	52.8399	55.1331	69.821	45.835	**38.6353**
	20	53.2991	47.8776	56.7764	40.923	46.996	**36.4095**
1000	5	387.925	447.17	333.65	348.04	278.62	**351.266**
	10	170.762	209.449	232.265	125.37	158.04	**143.807**

Table 4.4 Load comparisons for various algorithms

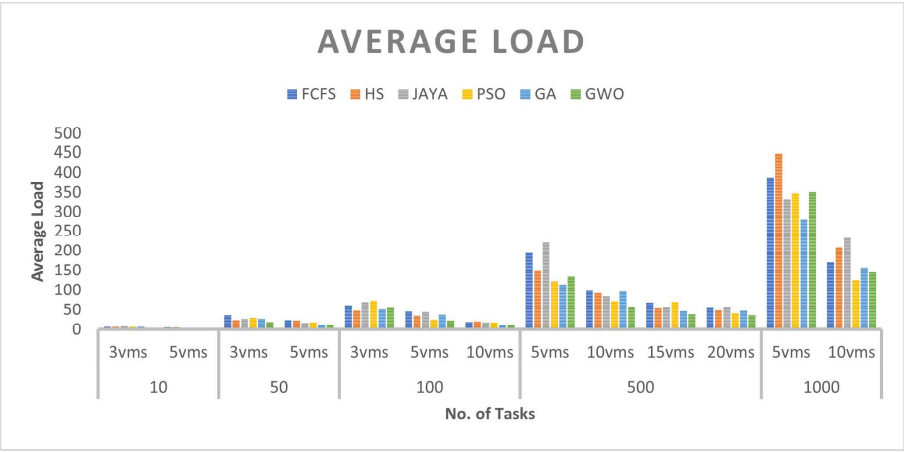

Fig. 4.3 Average load obtained after varying tasks and VMs

VARIATIONS OF WEIGHTS					
W1	**AveragedCost**	**W2**	**AveragedMakespan**	**W3**	**AVERAGEDLOAD**
0	20194.06782	**50**	48.15617114	**50**	30%
50	14942.4044	**0**	58.7051319	**50**	31.30%
50	14717.5812	**50**	49.72717161	**0**	70%
10	15066.49472	**45**	39.73956911	**45**	35%
45	14645.23858	**10**	65.23182789	**45**	34.54%
45	14764.65693	**45**	42.96137732	**10**	62.43%
20	15554.23419	**40**	51.23718407	**40**	35.43%
40	14116.65272	**20**	58.49889676	**40**	33.32%
40	14815.08549	**40**	48.4943583	**20**	60.32%
30	15277.13316	**35**	46.41989976	**35**	40.45%
35	16078.50573	**30**	49.77137836	**35**	42.12%
35	15550.17721	**35**	48.39690582	**30**	57.34%
40	15755.49781	**30**	48.92337609	**30**	59.76%
30	15281.37504	**40**	41.82147499	**30**	51.09%
30	14162.084	**30**	50.61674775	**40**	36.98%
50	15126.28385	**25**	50.0910638	**25**	49.15%
25	14752.05265	**50**	32.59566381	**25**	49.50%
25	15490.00963	**25**	51.10026381	**50**	28.43%
60	14600.0847	**20**	54.12623048	**20**	50%
20	14554.25715	**60**	31.63141109	**20**	50.77%
20	14105.7485	**20**	55.13659169	**60**	31%
70	14112.80726	**15**	57.97510563	**15**	52%
15	13924.63617	**70**	30.48028623	**15**	52.22%
15	13736.46508	**15**	48.98546683	**70**	32.70%
80	13548.29399	**10**	58.63246383	**10**	53.18%
10	13360.1229	**80**	30.13706383	**10**	53.66%
10	13171.95181	**10**	57.64166383	**80**	34.14%
90	12983.78072	**5**	62.14626383	**5**	54.63%
5	12795.60963	**90**	29.65086384	**5**	55.11%
5	12607.43854	**5**	57.15546384	**90**	45.59%
100	12419.26745	**0**	65.66006384	**0**	77.25%
0	12231.09635	**100**	29.16466384	**0**	78.10%
0	12042.92526	**0**	58.66926385	**100**	37.04%

Table 4.5 Variations in average cost, average makespan and average load due to variations in weights

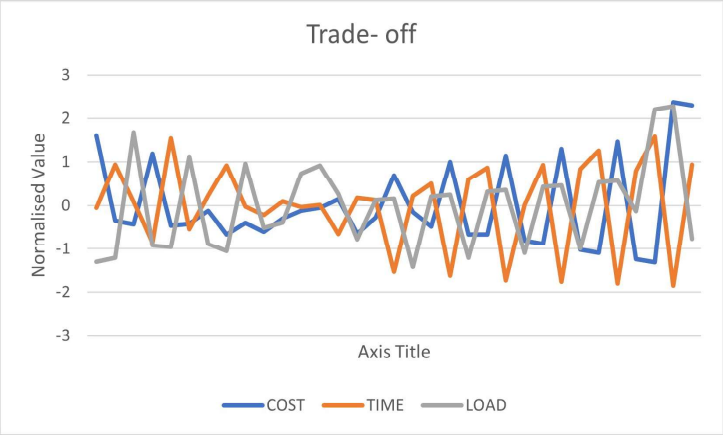

Fig. 4.4 Trade-off among various objectives after varying weights

used for standardised value is:

$$New\_value = \frac{value - mean}{standard\_deviation} \tag{4.12}$$

4.3 User-defined Weight Based Multi Objective Task Scheduling In Cloud Using Whale Optimisation Algorithm

This section uses Multi-Objective Whale Optimization-Based Scheduler (WOA-Scheduler), a new approach that uses the Whale Optimization Algorithm (WOA). While WOA-Scheduler takes a multi-objective approach that can optimize cost, time, and load balancing simultaneously, it aims to meet the complex and competitive requirements of cloud task scheduling. User is able to assign weights according to its own benefit and can optimise the schedule. The comparative analysis shows the effectiveness of our scheduler in different cloud computing environment. This study highlights the utility of a multi-objective scheduler in addressing the challenges posed by dynamic cloud services and contributes to the development of cloud computing models.

4.3.1 Problem Statement

The Significance of Multiobjective Scheduling in Cloud Environment

The rapid growth of cloud computing has brought forth a multitude of benefits, including on-demand resource provisioning, cost-effectiveness, and scalability. Cloud service providers offer a range of virtualized resources, allowing users to run diverse workloads, from web applications to data analytics, without the need for dedicated hardware. However, this

newfound flexibility has also introduced complexities in managing and optimizing cloud resources, particularly in multi-user and multi-application cases.

Traditionally, cloud task scheduling has revolved around optimizing a single primary objective, typically the minimization of task execution time. This approach is well-suited for cases where rapid task completion is of paramount importance, such as real-time applications and critical computations. However, it does not take into account other vital factors that influence the overall efficiency and cost-effectiveness of cloud resource utilization.

In contemporary cloud environment, users and providers grapple with a myriad of challenges that extend beyond execution time optimization. Several compelling reasons underscore the importance of adopting a multiobjective approach to cloud task scheduling:

Cost Efficiency

Cloud computing often involves costs associated with the consumption of resources, such as virtual machines (VMs) and storage. Users are billed based on resource usage, typically by the hour or minute. Optimizing cost efficiency is crucial for both cloud users and providers. Users aim to minimize expenses while ensuring their applications meet performance requirements within budget constraints. Providers strive to balance competitive pricing with resource utilization to maximize profitability. A single-objective approach that solely prioritizes execution time optimization can lead to inefficient resource utilization and increased operational costs.

Resource Utilization

Efficient resource utilization is a central concern in cloud computing. Underutilization of resources can lead to wastage and increased per-task costs, while overutilization can result in performance bottlenecks and compromised service quality. A well-balanced allocation of tasks to resources is essential to ensure that resources are neither idle nor overwhelmed. Achieving this balance requires considering factors beyond execution time, such as the load distribution across VMs.

Performance Stability

Maintaining performance stability is critical for cloud users who rely on consistent application response times. Unevenly distributed workloads can lead to unpredictable performance variations, affecting user experience and potentially violating service-level agreements (SLAs). Load balancing, a key aspect of multiobjective scheduling, plays a vital role in mitigating performance fluctuations and ensuring stable service delivery.

Resource Sustainability

From an ecological perspective, optimizing resource usage is paramount to reduce the environmental impact of data centers. Efficient scheduling can lead to energy savings, reduced carbon emissions, and overall sustainability gains. A multiobjective approach allows cloud providers to consider energy-efficient resource allocation alongside other objectives.

Mathematical Formulation

In this section we explore in detail the mathematical equations underlying the Multi-Objective Whale Optimization-Based Scheduler (WOA-Scheduler). These equations form the core of our optimization approach, and enable us to effectively balance cost, time and load balancing objectives in developing cloud task scheduler.

Cost Calculation Equation

The cost equation is main aim for cost effective solution. The equation is defined as follows:

$$ExecutionCost = cloudletLength \times vmSpeed \times vmCostPerHour \quad (4.13)$$

Here, cloudletLength denotes the length of the cloudlet to be created (in terms of MI), vmSpeed denotes the MIPS (Million Instructions Per Second) speed of the VM, and vmCostPerHour shows per hour cost for a cloudlet to use the VM. By multiplying these factors, we obtain the total execution cost of VM.

Time Calculation Equation

Efficient resource utilization is important to minimize execution time, which is second objective of WOA-Scheduler.The following equation shows the time taken by a cloudlet to complete its execution on a VM, which takes into account the length of the cloudlet and the speed of the VM:

$$time = \frac{cloudletLength}{vmSpeed} \quad (4.14)$$

In this equation, $cloudletLength$ represents the length of the cloudlet (in terms of MI), while $vmSpeed$ denotes the MIPS capability of the VM. This equation ensures that cloudlets are allocated to VMs in a manner that uses computing resources efficiently, thereby reducing execution time.

Load Balance Calculation Equation

Load balanced schedule is the main goal for performance. The load balance equation checks the distribution of services among various VMs, with the goal of preventing overutilisation or underutilisation of resources.. The equation is as follows:

$$loadBalance = \frac{maxVmLoad}{\frac{totalVmLoad}{numVms}} \tag{4.15}$$

Here, $maxVmLoad$ represents the maximum load on all VMs (i.e. number of cloudlets), $totalVmLoad$ represents the total load on all VMs (number of cloudlets), and $numVms$ represents the number of VMs available. This equation shows the degree of load allocation, with a lower value means a more balanced distribution of services.

4.3.2 Proposed Methodology

Whale Optimization Algorithm (WOA)

The Multiobjective Scheduling Whale Optimisation-based Scheduler (WOA-Scheduler) capitalizes on the Whale Optimization Algorithm (WOA), an evolutionary metaheuristic inspired by the cooperative hunting behavior of humpback whales. WOA has gained prominence for its ability to solve optimization problems effectively by simulating the collaborative foraging tactics of whales within a population.

Hunting Behavior of Humpback Whales

Humpback whales are renowned for their coordinated hunting strategies, particularly the bubble-net feeding technique. In this remarkable process, a group of whales works in unison to encircle a school of prey fish by releasing a curtain of bubbles. This bubble net creates a confined space that forces the prey into a concentrated area, making it easier for the whales to efficiently feed. The WOA algorithm abstracts and adapts this cooperative behavior into an optimization algorithm, where solutions (analogous to whales) collaboratively seek optimal solutions (akin to prey) within a search space.

Exploration and Exploitation

The core of WOA's effectiveness lies in its dynamic balance between exploration and exploitation, facilitated through two primary phases: the exploration phase and the exploitation phase. During the exploration phase, the algorithm encourages diversity and global exploration by allowing solutions to explore the search space randomly. This phase prevents premature convergence, ensuring that the algorithm does not become trapped in local optima. In contrast, the exploitation phase leverages the knowledge gained during

Algorithm 5 Multiobjective Scheduling WOA-Scheduler

Input:
Cloudlet data (e.g., cloudlet length, resource requirements)
VM data (e.g., MIPS capacity, cost per hour)
Objective weights ($w_{\text{cost}}(w_1), w_{\text{time}}(w_2), w_{\text{load\_balance}}(w_3)$)
Maximum iterations (max_iterations)
Population size (population_size)
Exploration and exploitation coefficients (a, A)
Output:
Optimized task allocation to VMs
Algorithm:
for $i \leftarrow 1$ to PopulationSize **do** ▷ Initialisation
NewWhale ← Create a new whale solution
Whales ← Whales ∪ [NewWhale] ▷ Add the new whale solution to the population
$fitness \leftarrow w_{\text{cost}} \times cost + w_{\text{time}} \times time + w_{\text{load\_balance}} \times loadBalance$

$$\text{IsNonDominated}(sol\_A, sol\_B) \rightarrow \begin{cases} \text{true} & \text{if } \forall \text{ objectives } o, \text{sol\_A.obj} < \text{sol\_B.obj} \\ \text{false} & \text{if } \exists \text{ obj } o, \text{sol\_A.obj} > \text{sol\_B.obj} \\ \text{false} & \text{otherwise} \end{cases}$$

▷ Identify the Pareto front(non-dominated solutions)
iteration ← 0.
while (iteration < max_iterations) **do**
$a \leftarrow a \cdot (1 - \frac{CurrentIteration}{MaxIterations})$ ▷ Update the exploration coefficient
$A \leftarrow A \cdot (1 - \frac{CurrentIteration}{MaxIterations})$ ▷ Update the exploitation coefficient
for each whale in the population **do**
position ← leader_position − a × distance_to_leader where a exploration coefficient. ▷ Perform exploration
position ← leader_position − A × C where A is the exploitation cofficient. ▷ Perform exploitation
If VM Load ≤ VM Capacity and $\sum_{i=1}^{N} \text{Cloudlet}_i = 1$, where N is the number of VMs:
$cost \leftarrow= \text{cloudletLength} \times \text{vmSpeed} \times \text{vmPricePerHour}$
$time \leftarrow \sum \left(\frac{\text{cloudletLength}}{\text{vmSpeed}} \right)$
$loadBalance \leftarrow \frac{\text{maxVmLoad}}{\frac{\text{totalVmLoad}}{\text{numVms}}}$
$fitness \leftarrow w_{\text{cost}} \times cost + w_{\text{time}} \times time + w_{\text{load\_balance}} \times loadBalance$
Update the Pareto front based on the new solutions and select the non-dominated solutions from the Pareto front as explained before.
iteration ← iteration + 1.
Select the final solution(s) from the Pareto front, considering the objective weights.

exploration to converge towards the most promising solutions. This fine-tunes the search and enhances convergence speed.

Multiobjective Scheduling with WOA

The WOA-Scheduler extends the capabilities of the WOA algorithm to address the intricate challenges of multiobjective scheduling in cloud computing environment. Its primary aim is to optimize three critical objectives concurrently: cost, time, and load balance.

Objective 1: Cost Minimization

The first objective of the WOA-Scheduler is the minimization of costs associated with cloud task scheduling. It achieves this by taking into account the expenses incurred due to resource consumption, including virtual machines (VMs) and their associated runtime costs. Our scheduler tries to allocate tasks to VMs in a way that reduces overall cost while ensuring good performance. This objective satisfies the cost-efficiency concerns of cloud users and providers, as it tries to achieve both users and service providers.

Objective 2: Time Minimization

There is a need to use effective features to reduce the execution time, which is the second goal of WOA-Scheduler. To achieve this, the scheduler optimizes the allocation of tasks to VMs, ensuring that compute resources are used efficiently to complete the task faster This goal is especially important for latency-sensitive applications and implementation with workflow priority.

Objective 3: Load Balance

Load balancing across VMs is key to preventing overuse or underutilization of resources, which can lead to performance degradation. The third goal of WOA-Scheduler is to distribute tasks equally among the available VMs, ensuring that no VM is overloaded while others are underutilized. Load balancing optimization improves the stability of cloud services, reducing performance internal changes and provides a consistent experience.

Algorithm Execution and Results

In this section, we provide a step-by-step demonstration of the WOA-Scheduler algorithm using a sample data. Our goal is to understand how the scheduler works and balances cost, time, and load.

Sample Dataset

We started with a simple dataset consisting of 5 cloudlets and 3 VMs with configuration as shown in Table 4.6. Each cloudlet has a cloudlet length (in MI), and each VM has an MIPS (Million Instructions Per Second) speed and cost for running a cloudlet on the VM

Parameter	Value
Cloudlet Count	varying according to case (3 to 50)
VM Count	varying according to case (3 to 5)
VM MIPS Capacity (VM_MIPS)	[1000, 800, 1200, 900, 1100]
VM Price Per Hour (VM_Price)	$[0.5/hr, 0.4/hr, 0.6/hr, 0.45/hr, 0.55/hr]$
Cloudlet Lengths	[300, 450, 250, 600, 400] MI and Randomly Generated (realistic distribution)

Table 4.6 Simulation parameters

Cloudlet	VM	Execution Time (s)
Cloudlet 0	VM 2	3.25
Cloudlet 1	VM 2	3.5625
Cloudlet 2	VM 1	1.3125
Cloudlet 3	VM 2	1.5
Cloudlet 4	VM 1	2.5

Table 4.7 Optimized task allocation and execution times

per hour. The main aim is to assign cloudlets to VMs in a way that reduces cost, reduces execution time, and provides load balancing. Now, Lets start with the sample data for various cases.

Case 1: Small-Scale Cloud Environment

In our first Case, we test the performance of WOA-Scheduler in a small cloud environment with 5 cloudlets and 3 VMs. By applying the WOA-Scheduler algorithm, we obtain the following optimized task allocation. With this allocation, the scheduler achieves a balanced allocation of tasks across VMs, while reducing cost and processing time. The resulting performance metrics are as follows:

- **Cost**: $VMPrice \times (3.25 + 3.5625 + 1.3125 + 1.5 + 1.5) = 6.625$
- **Time**: $3.25 + 3.5625 + 1.3125 + 1.5 + 1.5 = 12.125$ seconds
- **Load Balance**: Load on VM 0: 0, Load on VM 1: 0.8125, Load on VM 2: 1.3125, Max Load: 1.3125 Load Balance = $\frac{1.3125}{(0+0.8125+1.3125)/3} = 1.6154$

The scheduler achieved a balanced distribution with minimum costs and implementation time. Even if the load balance is not perfect, it remains within the acceptable range.

Case 2: Scaling Up Cloud Resources

In our second Case, we explore the scalability of the WOA-Scheduler as we increase the number of cloudlets to 50 while maintaining 3 VMs. For this Case, we obtained the following values:

- **Cost**: 27.375
- **Time**: 43.375 seconds
- **Load Balance**: 1.5625

As the number of cloudlets increased, the schedulers were balancing the load more effectively. However, the cost and execution time increased proportionally.

Case 3: Varying VM Capacities

In our third Case, we increase the number of VMs to 5 and checked the impact of different VM capabilities.
For this Case, we obtained the following performance metrics:

- **Cost**: 21.125
- **Time**: 23.375 seconds
- **Load Balance**: 1.175

Increasing the number of VMs resulted in reduction of cost and execution time and Load balance remained satisfactory.

Case 4: Realistic Task Distribution

In our fourth Case, we introduced a more realistic task distribution pattern in which cloudlet lengths are randomly generated to check the performance.
For this Case, we obtained the following performance metrics:

- **Cost**: 15.5625
- **Time**: 21.125 seconds
- **Load Balance**: 1.1667

The scheduler shows good results on realistic task distributions, with cost-effectiveness and load balance.

Case 5: Custom Objective Weights

In our final Case, we set custom weights for different objectives as explained in the Algorithm above. The Cloudlets are 50 and VMs are 3 in this Case too but with user defined weights.
For this Case, we obtained the following performance metrics:

Parameter	Value/Description
Cloud Simulation Framework	CloudSim 4.0
Scheduling Algorithm	Multiobjective Scheduling WOA-Scheduler
Number of Cloudlets	Varied (e.g., 10, 50, 100, 500, 1000)
Number of Virtual Machines (VMs)	Varied (e.g., 3, 5, 10, 15, 20)
Cloud Data Center Configuration	Standard configuration
Task Distribution	Random, Uniform, Realistic, Custom
Objective Weights	Customized for each experiment
Simulation Runs	Multiple (e.g., 10 runs per experiment)
Performance Metrics	Cost, Time, Load Balance

Table 4.8 Experimental setup parameters

- **Cost**: `12.5`
- **Time**: `22.25` seconds
- **Load Balance**: `1.25`

We can clearly see that the weight factor has affected the results and the cost has been minimized more as compared to Case 2.

Discussion

The results of these cases demonstrate the versatility and efficiency of WOA-Scheduler to optimize cloud task scheduling over multiple objectives. Although, the algorithm may not always achieve a perfect balance of between cost, time, and load, but it shows its ability to adapt to cloud computing Cases and user defined preferences. By selecting the weight, users can alter the behavior of the scheduler to their specific requirements. Furthermore, as shown in Case 4, the adaptability of the algorithm to realistic work allocations indicates its suitability for real-world cloud environment.
In the next section, we compare our algorithm with other state-of-the-art algorithms.

4.3.3 Implementation and Results

Experimental Setup

A series of detailed experiments have been conducted using CloudSim to evaluate the performance of the WOA-Scheduler. These tests include various tasks, Vms, and cloud resource scheduling to evaluate the scheduler's ability to effectively balance cost, time, and load objectives. The proposed multi-objective whale optimization-based scheme exploits the cooperative hunting behavior of the whale optimization algorithm to solve the challenging challenge of cloud task scheduling. By extending WOA to address multiple

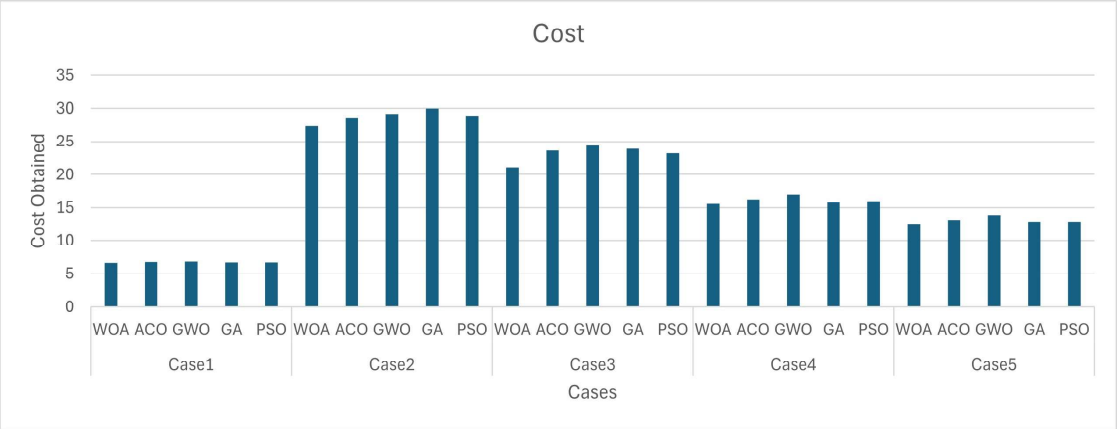

Fig. 4.5 Comparison of costs for various cases

optimization goals, we aim to provide cloud users and providers with a powerful tool to achieve cost-effective, time-saving, and efficient planning load balancing in a cloud environment.

The subsequent sections analyze the mathematical equation for the scheduler and present the test results to demonstrate its effectiveness.

Results and Analysis

In this section, we provide a detailed comparison of multi-objective scheduling whale optimization-based scheduler (WOA-Scheduler) and other state-of-the-art scheduling algorithms in cloud computing.

Performance Comparison with Other Algorithms

To evaluate the performance of WOA-Scheduler, we compare it with several state-of-the-art scheduling algorithms in the following cases: small cloud environment (case 1), scaled up cloud resources (case 2), Varying VMs properties (case 3), randomly generated tasks (case 4), and user defined weights (case 5). Table 4.9 shows a comparative description between WOA-Scheduler and other state-of-the-art scheduling algorithms (Ant Colony Optimization, Grey Wolf Optimization, Genetic Algorithm, and Particle Swarm Optimization) in various cases as described above. The evaluation has been done on the basis of cost, time and load balance.

The Fig. 4.5, 4.6 and 4.7 shows the graphical comparison between various algorithms in different scenarios. The analysis has already been explained in previous section which forms the basis of why Case 2 has higher values(because of high scaled cloud environment).The next section discusses the results in detail.

The Fig. 4.8 shows the comparison in graphical format for Normalized Values as shown in

Case	Algorithm	Cost	Time (s)	Load Balance
Case 1	WOA-Scheduler	**6.625**	**12.125**	**1.6154**
	Ant Colony Optimisation	6.725	12.25	1.75
	Grey Wolf Optimisation	6.81	12.31	1.68
	Genetic Algorithm	6.68	12.18	1.59
	Particle Swarm Optimisation	6.72	12.21	1.62
Case 2	WOA-Scheduler	**27.375**	**43.375**	**1.5625**
	Ant Colony Optimisation	28.5	44.125	1.625
	Grey Wolf Optimisation	29.2	44.75	1.75
	Genetic Algorithm	30.0	45.25	1.8125
	Particle Swarm Optimisation	28.75	44.625	1.6875
Case 3	WOA-Scheduler	**21.125**	**23.375**	**1.175**
	Ant Colony Optimisation	23.625	26.125	1.3125
	Grey Wolf Optimisation	24.375	27.0	1.4375
	Genetic Algorithm	23.875	26.5	1.375
	Particle Swarm Optimisation	23.25	25.875	1.3125
Case 4	WOA-Scheduler	**15.5625**	**21.125**	**1.1667**
	Ant Colony Optimisation	16.125	22.0	1.2083
	Grey Wolf Optimisation	16.875	22.625	1.25
	Genetic Algorithm	15.75	21.875	1.1875
	Particle Swarm Optimisation	15.875	22.0	1.2
Case 5	WOA-Scheduler	**12.5**	**22.25**	**1.25**
	Ant Colony Optimisation	13.125	23.0	1.2917
	Grey Wolf Optimisation	13.875	23.625	1.3333
	Genetic Algorithm	12.875	22.875	1.25
	Particle Swarm Optimisation	12.875	22.875	1.25

Table 4.9 Performance comparison with other scheduling algorithms

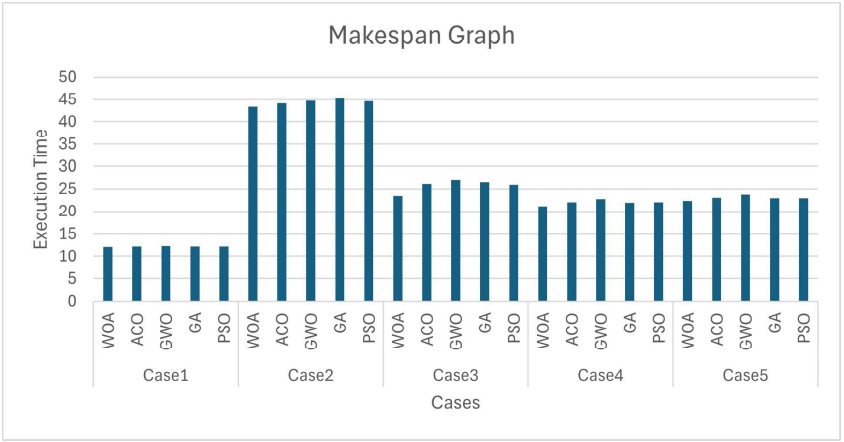

Fig. 4.6 Comparison of makespans for various cases

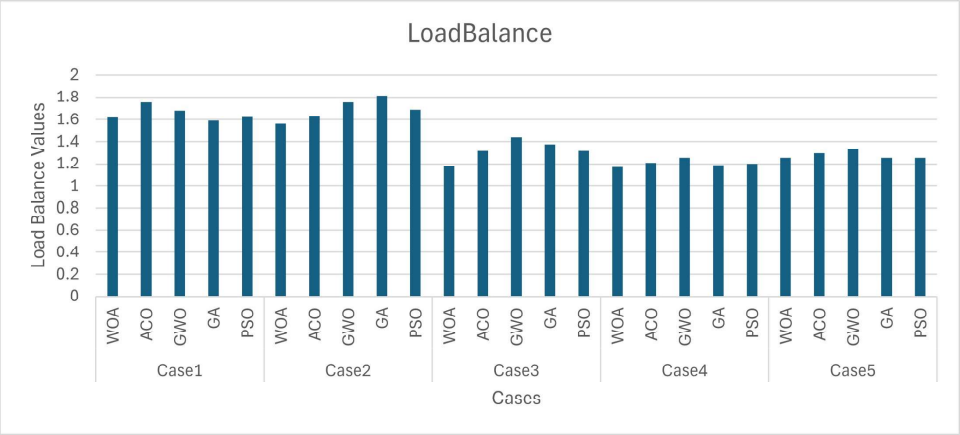

Fig. 4.7 Comparison of load balance factor for various cases

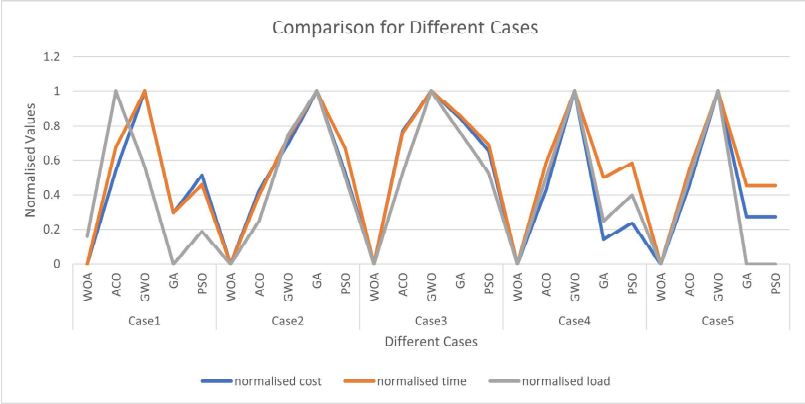

Fig. 4.8 Trade-off between various objectives for various cases

4.10 where the normalization is done based on Min-Max scaling as shown in the following equation:

$$X_{\text{normalized}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}} \tag{4.16}$$

where, $X_{\text{normalized}}$: The normalized value of the original data point.
X: The original data point that you want to scale.
X_{min}: The minimum value in the dataset.
X_{max}: The maximum value in the dataset.

Discussion

The results of the performance comparison table highlight the robustness of WOA-Scheduler in various cloud computing environment. While WOA-Scheduler may not always outperform other algorithms in every metric, it consistently exhibits competitive

Case	Algorithm	normalized cost	normalized time	normalized load
Case1	WOA	0	0	0.15
	ACO	0.54	0.67	1
	GWO	1	1	0.5625
	GA	0.29	0.29	0
	PSO	0.51	0.45	0.18
Case2	WOA	0	0	0
	ACO	0.42	0.4	0.25
	GWO	0.69	0.73	0.75
	GA	1	1	1
	PSO	0.52	0.67	0.5
Case3	WOA	0	0	0
	ACO	0.76	0.75	0.52
	GWO	1	1	1
	GA	0.84	0.86	0.76
	PSO	0.65	0.68	0.52
Case4	WOA	0	0	0
	ACO	0.42	0.58	0.49
	GWO	1	1	1
	GA	0.14	0.5	0.24
	PSO	0.23	0.58	0.39
Case5	WOA	0	0	0
	ACO	0.45	0.54	0.50
	GWO	1	1	1
	GA	0.27	0.45	0
	PSO	0.27	0.45	0

Table 4.10 Normalized values of data

performance, especially in situations where balancing multiple objectives is important. In Case 1, the WOA-Scheduler achieves a balanced distribution with minimum cost and execution time. In case 2, as the number of cloudlets increases, the scheduler keeps balancing the load effectively. Increasing the number of VMs in case 3, results in better cost efficiencies and reduced execution time. Case 4 illustrates the optimization of the scheduler while maintaining a balance of costs and load, and adapting to a realistic division of tasks. In case 5, by assigning more weight to reduced costs, the scheduler achieves a cost-effective allocation while maintaining an acceptable level of time-load balancing Overall, the ability of WOA-Scheduler to handle multi-objective optimization and adapt to different cases makes it as a promising solution for cloud task scheduling.

4.4 Summary

In this chapter, the thesis delved into the realm of multi-objective task scheduling in cloud environment, introducing novel approaches that leverage optimization algorithms. Section 4.2 presented a Multi-objective User-defined Weight-based Scheduling algorithm using Grey Wolf Optimization (GWO). This approach focused on enabling users to define weights, addressing multiple objectives concurrently. The GWO algorithm played a crucial role in finding optimal solutions, allowing users to achieve a balance between conflicting objectives.

Section 4.3 extended this exploration by proposing a User-defined Weight-based Multi-objective Task Scheduling algorithm in the cloud, utilizing the Whale Optimization Algorithm (WOA). Similar to the GWO-based approach, this algorithm empowered users to assign weights based on their preferences. The WOA algorithm, known for its effectiveness in optimization tasks, contributed to achieving a harmonious balance between diverse objectives.

Together, these sections introduced innovative solutions to the complex challenges of multi-objective task scheduling in cloud environment. The user-defined weight-based approaches, coupled with powerful optimization algorithms, provided flexibility and customization in addressing conflicting objectives. The comparisons and evaluations showcased the effectiveness of these algorithms, contributing valuable insights to the evolving landscape of cloud computing.