International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(10s) 2026 pp. 227-244
https://doi.org/10.70917/ijcisim-2026-3580
Article

DWCSO: A Hybrid Intelligent Algorithm for Multi-Type Workload Scheduling in Cloud Computing

Jai Bhagwan[1], Seema Rani[2*], Sanjeev Kumar[1], Sunila Godara[1]
[1]Department of Computer Science & Engineering, Guru Jambheshwar University of Science & Technology, Hisar, India.
Email: drjaicse@gmail.com, ORCID: 0000-0002-8708-4029.
[2*]Department of Computer Science & Engineering, Ch. Devi Lal State Institute of Engineering & Technology, Sirsa, India.
Email: seemarajput1028@gmail.com.

Abstract: Cloud computing is the emerging technology used by almost every industry in the era of artificial intelligence. Small to large organizations are continuously moving towards cloud-based systems for ensuring easily availability of data, fast computation and other pay-per usage IT sector services in remote areas. As a result the workload of cloud data-centers is increasing regularly as the cloud technology provides fast computational pay-per-use services. So, resource management and scheduling have been vital issues for research since last decade. The data-centers need to be maintained at various stages like network, cooling systems, servers' maintenance etc. One of the major mechanisms of resource management is task scheduling. Various task scheduling techniques using swarm intelligence have been developed so far. However, these techniques suffer from either exploration or exploitation. So, this paper proposes a hybrid DWCSO tasks scheduling algorithm which is based on Differential Evolution (DE), Adaptive Inertia Weight (SMIW) and Cat Swarm Optimization algorithm. The SMIW inertia weight balances the seeking and tracing modes. The DE based operators help the DWCSO algorithm to escape from the local stagnation. The proposed algorithm has been tested on various scientific workflows and Google Traces 2019 independent-tasks datasets. After evaluation, it is observed that the proposed DWCSO algorithm consistently outperforms its competitor IGWO, achieving 9.41 to 17.33% improvement in makespan, 4.93 to 12.56% in cost with higher throughput in case of workflows. In case of Google Traces 2019, the proposed DWCSO achieves 15.52 to 19.32% improvement in makespan, 8.05 to 10.20% in cost over IGWO with higher throughput. The maximum improvement observed against PSO is demonstrating its superior efficiency.
Keywords: Cloud Computing, Cat Swarm Optimization (CSO), DWCSO, Independent Workload, Workflows Workload, SMIW, Virtual Machines (VMs).

1. Introduction

Cloud computing has arisen as a dominant paradigm for delivering computing resources such as storage, processing power, and software services over the internet. Its scalability, flexibility, and cost-effectiveness make it suitable for handling large-scale and data-intensive applications. However, efficient resource management remains a major challenge, where task scheduling plays a crucial role in improving system performance and user satisfaction [1, 2]. Task scheduling is the process assigning tasks to virtual machines in a host in such a way that optimize several parameters like makespan, execution cost, energy consumption, and resource utilization. Task scheduling is considered an NP-hard optimization problem, due to the heterogeneous and dynamic nature of cloud environments. It is difficult to obtain optimal solutions using traditional methods, nowadays in artificial intelligence era [3, 4]. Initially, classical algorithms such as Min-Min, Max-Min, and FCFS were used for scheduling. But, these approaches fail to adopt dynamic workloads and multi-objective requirements. So, to overcome these boundaries, scientists have developed swarm intelligence algorithms such as Grey Wolf Optimization (GWO), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), Cat Swarm Optimization (CSO), Whale Optimization Algorithm (WOA) [5, 6] and others. In recent years, various hybrid optimization techniques have been developed as they combine the strengths of multiple algorithms to improve convergence speed and avoid local optima.

These methods have shown promising results in optimizing multiple Quality of Service (QoS) parameters simultaneously, including cost, makespan, throughput [7, 8, 29] etc.

Despite significant advancement, several challenges remain unresolved such as many existing

approaches rely heavily on small scale data simulations that do not fully reflect real-world cloud environments. Additionally, various issues for example scalability, multi-type workload handling, and considering many objectives optimization still require more research. Therefore, developing efficient and considering large scale and multi-type tasks scheduling algorithms continues to be a hot research path in cloud computing systems [9, 10, 19].

1.1 Contribution of Paper

The main contributions of this paper are:

- Addressing multi-type workload (scientific workflows and Google Traces 2019 independent workload) scheduling to minimize makespan and cost while simultaneously improving system throughput.
- Introduced HEFT policy for workflows and independent workloads initialization on virtual machines.
- Used a self-motivated inertia weight (SMIW) to balance seeking and tracing modes of CSO algorithm.
- Designed a hybrid DWCSO algorithm by injecting differential evolution and SMIW in CSO algorithm. The DE method has been used for better exploration i.e. discovering better solutions.
- Compared DWCSO algorithm with recently developed IGWO [28] in 2025 and other algorithms developed time to time. The DWCSO algorithm's has acquired superior performance.

1.2 Paper Organization

The rest of the paper is organized as: section 2 explains the literature review, in section 3, system model and problem definition are addressed, section 4 presents algorithm design, section 5 presents the results and discussion and finally conclusion and future directions are suggested in section 6.

2. Related Work

Task scheduling in cloud computing has been widely explored using swarm intelligence and hybrid optimization techniques to improve performance of the system by optimizing makespan, cost, throughput, response time, energy consumption etc. In [1], a Modified Grey Wolf Optimization algorithm was proposed to enhance task scheduling by incorporating a mean-based mechanism in the hunting process. The method's main focus was on minimizing makespan and energy consumption. The proposed method demonstrated improved convergence compared to PSO and standard GWO algorithms. A hybrid PSO-GWO approach was presented in [2] for workflow scheduling, where PSO was used for global searching and GWO for local exploitation. The proposed algorithm optimized execution cost and total execution time by achieving better performance than standalone PSO and GWO algorithms. The research in [3] introduced a hybrid CSO with Tabu Search to avoid local optima problem. It demonstrated significant performance improvement compared to PSO and FCFS. In [4], a Rider Cuckoo Optimization Algorithm was proposed to improve task scheduling efficiency. The main target of the algorithm was makespan, energy consumption, faster convergence and better optimization performance compared to PSO and Cuckoo Search. A prioritized task scheduling approach based on Cat Swarm Optimization was proposed in [5], where tasks and VMs prioritization methods were designed. The CSO method with priority mechanisms improved SLA violation rate, energy consumption and makespan. In [6], an enhanced PSO-based scheduling algorithm was developed by enhancing particle initialization and search methods. The proposed approach focused on execution time, makespan reduction and improving convergence speed. In [7] a hybrid Genetic Algorithm was designed to address multi-objective scheduling issues. The proposed algorithm optimized makespan, cost and resource utilization in a heterogeneous cloud environment. A hybrid Decision Tree and GWO algorithm was designed in [8]. The decision tree was used to classify the tasks before scheduling to improve load balancing, cost and execution time. In [9] a parallel enhanced Whale Optimization Algorithm was introduced for independent tasks scheduling. The approach improved makespan, scalability and convergence speed specially in a large scale environment.

A novel hybrid algorithm called GA-GWO was designed in [10] to address scheduling in a heterogeneous environment. The algorithm efficiency minimized makespan, cost and energy consumption and demonstrated strong multi-objective optimization strength. A CHPSO algorithm was developed [11] using probabilistic workload modeling. The proposed approach improved response time, resource utilization and energy efficiency using dynamic adjustment of scheduling decisions. A hybrid intelligent algorithm was designed using Flower Pollination and GWO in [12] for scheduling workflows efficiently. The population

was initialized by PEFT algorithm. The novel hybrid algorithm FPAGWO was compared with FPAGA and found better. In [13] a hybrid version of GWO was introduced using Simulated Annealing to improve the load balance, cost efficiency and execution time under QoS constraints. To focus on heterogeneous cloud environment a mathematical optimization-based model was designed in [14]. The optimal solution was found by minimizing cost considering deadline and security constraints. In [15], a Genetic Algorithm was developed with a selective repair method for deadline-constrained scheduling. The method achieved cost minimization under deadlines satisfaction by repairing of infeasible solutions. A Synergistic Swam Optimization method was developed by injecting advanced search mechanisms [16] to improve scheduling efficacy. The proposed algorithm mainly focused on resource utilizing and execution performance optimization compared to other methods.

In [18], a priority based Cat Swarm Optimization algorithm was developed to address migration time, cost, makespan and resource utilization. The algorithm was found better than PSO and CS algorithms. In [28] research, the scientists designed a novel hybrid algorithm HEFT-IGWO using HEFT and differential evolution algorithm. This novel hybrid algorithm compared with GWO and HEFT and achieved improved performance in terms of makespan, cost and energy consumption. The scientists in [20] introduced a hybrid DECSO algorithm combining Differential Evolution and CSO. The method enhanced the balance between exploration and exploitation, optimizing makespan, migration time, and resource utilization.

2.1 Research Problem and Gaps

The recent studies [20] and [28] in 2025 are implemented using swarm intelligence and the research [20] did not consider workflows scheduling which is essential in the era of Artificial Intelligence as the small and large scale workflows are generated using IoT and other devices for computation. The second study [28] implemented at a small scale for workflows only and did not consider multi-type workloads.

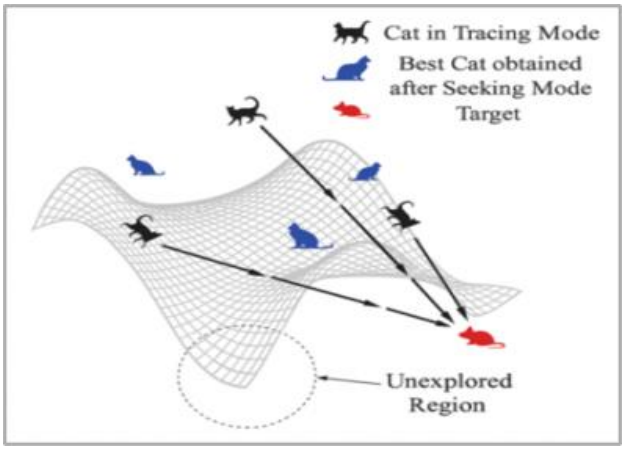

Fig. 1: Unexplored area in CSO

Furthermore, every swarm intelligence algorithm has issues in either exploration or exploitation phase. The CSO algorithm performs better for task scheduling than ACO, PSO and some other swarm intelligence algorithms [5, 17, 18, 25]. Still, the cat swarm optimization has the issue of exploration. So, potential optimal solutions may be missed if these are available in unexplored areas. Also, it may get paralyzed in exploitation due to lack of balance between seeking and tracing modes. The CSO [5, 18, 20, 22] algorithm has various limitations given as:

- The main issue is related to exploration as shown in Fig. 1 [22]. So, it gets trapped in seeking mode i.e. exploration which leads to premature convergence.
- A lack of balance between seeking and tracing modes is existing, due to that premature convergence problem may occur.
- More cats are sitting in seeking mode, so tracing mode may get paralyzed for better exploitation.

Additionally, the recent variants of CSO discussed in the literature are priority based CSO [5] and recently developed DECSO [20]. These algorithms are better than basic CSO [21] algorithm for task scheduling, but these are not considering workflows scheduling. So, a novel hybrid DWCSO algorithm has been proposed to overcome the problems discussed above.

3. System Model And Problem Definition

This section explains the system model and problem definition in detail. We have considered a data-

center having one host and 60 virtual machines for tasks scheduling in cloud computing. Various symbols used in this research paper are summarized in Table 1.

Table 1: Symbols and Meaning

Symbol	Meaning
DC	Datacenter
HM	Host Machine
VMs	Virtual Machines
OMap	Optimal Mapping
EST	Estimated Start Time
PT	Processing Time
SRD	Seeking Range Dimension
MR	Mixing Ratio
CDC	Count Dimensions to Change
SMP	Seeking Memory Pool
MIPS	Million Instructions Per Second
MI	Million Instructions
w_{SC}	VM Scaling Weight
F	Mutation Factor
CR	Crossover Rate
FT	Finishing Time
CF	Cost Factor
MF	Migration Factor
Migs	Number of Tasks Migrations
VM_u	Number of VMs Used

3.1 Datacenter Model

The data-center having host machines are modeled in (1).

$$\mathrm{DC} = \{\mathrm{HT1}, \mathrm{HT2}, \mathrm{HT3}, \ldots \mathrm{HTn}\} \quad (1)$$

Each host machine (HM) has various VMs as given in (2).

$$\mathrm{HM} = \{\mathrm{VM1}, \mathrm{VM2}, \mathrm{VM3}, \ldots \mathrm{VMm}\} \quad (2)$$

The virtual machines used in this research are groups of 5 types having lower to highest computing power between 500-1000 MIPS, RAM between 512-1024 MB and Bandwidth 1000 Mbps. Further, the detail is mentioned in Table 2.

Table 2: VMs Properties

VM Properties	
Parameter	**Value**
No. of VMs	60
Computing Power	500-1000 MIPS
RAM	512-1024 MB
Bandwidth	1000 Mbps
PEs (CPUs)	1 For Each VM

3.2 Problem Definition

We Consider a scheduling problem with a set of workloads W = {1, 2, 3, …, N} and a set of virtual machines (VMs) VS = {1, 2, 3, …, M} inside a single host and a single data-center. Every task $T_i \in W$ should be assigned to a $VM_j \in VS$ and the parent-child dependency should be maintained in case of workflows as these are directed acyclic graph structures that means no child task can be executed before a parent. Our objective is to determine an optimal tasks VMs mapping OMap(i) that minimizes the overall makespan, and computation cost, and improve throughput simultaneously.

3.3 Makespan Model

The makespan [19] i.e. total schedule length is given in (3).

$$makespan = max(FT_i) \tag{3}$$

Where, i∈W, W is set of all tasks or workload and FT_i is finishing time of tasks i.

3.4 Cost Model

In real cloud computing environment, a cloud service provider operates on pay-per-usage model. We have designed the cost model [12, 19] to keep in mind this pay-per-usage concept. It is assumed a data-center has one host containing several VMs utilizes the computation cost as given in (4).

$$cost = \frac{CF+MF}{2} \tag{4}$$

The Migration factor is calculated using (5). Let: HM is number of host machines, DC is number of data-centers, Migs is number of migrations of tasks and VM_u is number of VMs used, then:

$$MF = \frac{DC}{HM} \times \frac{Migs}{VM_u} \tag{5}$$

Computation cost factor is calculated using (6). Let: V is total number of VMs, PT_j is processing time on VM_j, Mem_j is total task memory on VM_j, $MIPS_j$ is MIPS of VM_j and DC_{TMips} is total data-centers MIPS, then:

$$CF_{base} = \frac{1}{V}\sum_{j=1}^{V} \frac{PT_j \times Mem_j}{MIPS_j \times DC_{TMips}} \tag{6}$$

VM scaling factor is used to scale up the cost if the number of VMs is increased using pay-per-usage model, and it is calculated using (7). Let: w_{sc} is VM scaling weight, then:

$$SF = 1 + w_{SC} \times V^2 \tag{7}$$

Final CF (cost factor) is calculated using (8). Let: norm is normalization factor i.e. 1000 to balance the cost for realistic environment.

$$CF = norm \times CF_{base} \times SF \tag{8}$$

3.5 Throughput

The throughput [29] metric is used to observe the system's performance and it is calculated using (9). The throughput metric has not been included in the fitness function and it has been calculated separately to avoid complex fitness.

$$throughput = Total\ Tasks/Makespan \tag{9}$$

3.6 Fitness Function

Fitness function (F_x) is a backbone of the optimization problem which is discussed in (10). We are keeping it normalized using the number of DC and VMs and minimizing the fitness function value in this research. The objective of this research is to optimize makespan and cost. So, the makespan and cost are summed up and multiplied with weights to give the special treatment to both. In this research makespan and cost are minimized by the algorithm during all generations.

$$F_x = \frac{w_1 \times makespan + w_2 \times cost}{DC \times VMs} \tag{10}$$

Where $w_1$and $w_2$weights' values are 0.6 and 0.4 respectively given that $w_1 + w_2 = 1$. Here, slightly more priority is given to makespan compared to the cost so that the system's performance could be given more focus rather than the cost.

4. Proposed Algorithm Design

This section presents the proposed algorithm and its detailed description.

4.1 Cat Swarm Optimization Theory

The Cat Swarm Optimization algorithm was developed in [21] by Chu et al. in year 2006. It uses seeking and tracing modes for exploration and exploitation of targeted food. In seeking mode the cats take rest and stay alert whereas in tracing mode, the cats exploit the target quickly [20, 22].

1) Seeking Mode

In this mode, basically four parameters are used: seeking memory pool (SMP), seeking range dimension (SRD), count of dimensions to change (CDC) and self-position consideration (SPC). To distribute the cats in

seeking and tracing modes, a mixing ratio (MR) is used. SM can be explained by the following steps:

- Make SMP copies of the K^{th} cat.
- Considering CDC, perform the following steps:
- Update the position randomly based on the SRD value as given in (11).

$$X_{K,D} = (1 + rand \times SRD) \times X_{K,D} \tag{11}$$

Where, $X_{K,D}$ is the position of the cat and rand is a random number between (0, 1).

- Discover the best cats by fitness calculation.
- Replace the original cat with best copy.

2) Tracing Mode

During this mode [5, 19- 22], the cats move fast towards its food by updating their velocities and positions i.e. quick exploitation. The steps involved in this mode are:

- Compute Cat_K velocity and position using (12).

$$V_{K,D} = V_{K,D} + \left(c1 \times r1 \times \left(X_{BEST,\ D} - X_{K,D}\right)\right) \tag{12}$$

Where, $V_{K,D}$ is velocity, c1 is acceleration coefficient and r1 is a random number between (0-1) in the basic CSO. We are using static inertia weight ω as 0.5 for basic CSO algorithm.

- Update the position of the Cat_K by the following (13).

$$X_{K,D} = X_{K,D} + V_{K,D} \tag{13}$$

Where, $X_{K,D}$ is the position of the Cat_K in dimension *D*.

4.2 Self-Motivated Inertia Weight

To balance the exploration and exploitation i.e. seeking and tracing modes of CSO, we have introduced a non-linear Self-Motivated Inertia Weight (SMIW). The SMIW inertia weight and other adaptive inertia weight strategies are suggested in [19, 23-25]. The mathematical model of the SMIW (ω) inertia weight for this research is given in (14).

$$\omega = \omega_{max} \times \exp\left(-l \times \left(\frac{itr}{itr_{max}}\right)^{g}\right) \tag{14}$$

Where, ω_{max} is the initial inertia weight set to 2.0. l is the local searching attractor and g is the global searching attractor. The local and global attracters are used to balance the local and global searching respectively. Basically g controls the decay rate with the iterations gradually. Larger value of g *than* l leads to exploitation and smaller value leads to exploration. So, changing of ω over iterations is crucial. Here, ω decreases with iterations increments as ω formula defines the exponential decay function. l and g are set to 2.0 to focus on balancing the exploration and exploitation. This inertia weight avoids premature convergence as it adaptively controls the velocity of CSO algorithm and makes the balance between exploration and exploitation. The adaptive inertia weights get decreased gradually during the time intervals as shown in Fig. 2. It means in initial iterations, the exploration of the solutions takes place as bigger inertia weight generates diverse populations. In the last iterations exploitation takes place i.e. best solutions are provided using eminent VMs considering makespan and cost. The example of the behaviors of inertia weights are displayed in Fig. 2. The Beta component of honey badger algorithm [27] is used to balance the honey badger positions update and it is picked up to show the behavioral example only. The S-type behavior is considered best for exploration and exploitation which is shown by SMIW inertia weight. So, in this research it is used to make the balance between seeking and tracing modes of our proposed algorithm.

After using the SMIW inertia weight the velocity update formula of tracing mode of the DWCSO becomes as shown in (15).

$$V_{K,D} = \omega \times V_{K,D} + \left(c1 \times r1 \times \left(X_{BEST,\ D} - X_{K,D}\right)\right) \tag{15}$$

But the position update formula will remain same as shown in (13).

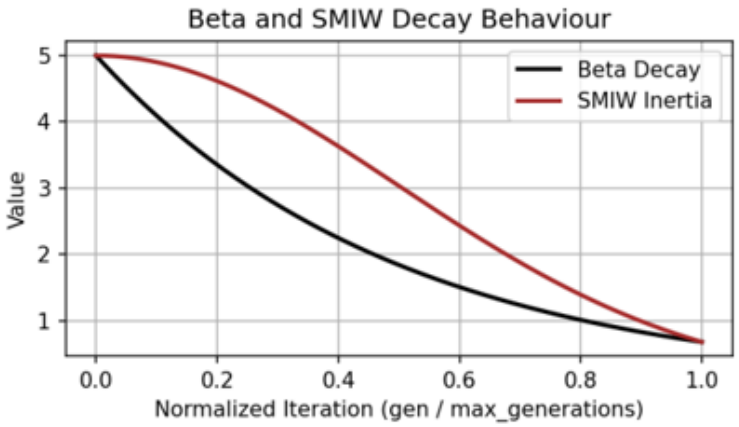

Fig. 2: Decay Example of SMIW and Beta Inertia Weights

4.3 HEFT Theory

The Heterogeneous Earliest Finished Time algorithm [26, 30] is very famous to estimate the earliest finished time and maintain the DAG dependency as well. The proposed HEFT algorithm can work for both dependent and independent tasks. In case of independent tasks, the dependencies are set 'nil' and the average execution time is calculated for the ranking i.e. priorities purposes, whereas in case of workflows the dependencies are included for scheduling as shown in Algorithm 1.

Algorithm 1: HEFT Algorithm

Input: Workloads T (Workflows or Independent Tasks), VMs V
Output: Initial Tasks Assignment to VMs

```
For each task t in T Do
    avg ← average(ET[t, vm] for all vm in V)
    If workflow Then
        rank(t) ← avg + max(rank(successors(t)))
    Else
        rank(t) ← avg
    End If
End For
Sort the tasks in descending order  // maintain critical task priory
For each solution S Do
    Set all VMs v finish times to 0
    For each task t in sorted list Do
        For each vm in V Do
            Est ← max(finish time of predecessors)
            ft ← Est + ET[t, vm]
        End For
        Choose vm with minimum finish time ft
        Assign task t to vm and update finish time ft
    End For
End For
return initialized population (initial tasks assignments to VMs)
```

4.4 Proposed DE Mutation Theory

To maintain solutions diversity and refined searching for more new solutions, differential evolution's (DE) mutation and crossover operators [20] are used in this research. The algorithm in displayed in Algorithm 2. The proposed DWCSO uses this DE phase algorithm only if it provides better solution during all generations. The working of the algorithm is as:

- Pick three different individuals: r1, r2, r3 to create variations.
- To introduce diversity and exploration use mutation given in (16).

$$\text{mutant} = x_1 + F \times (x_2 - x_3) \qquad (16)$$

- For crossover, take value from mutant with probability CR for each dimension, otherwise keep

original cat value as trial solution.

- Evaluate the fitness of trial and if the trial is better than current cat (solution), replace it otherwise keep original cat.

Algorithm 2: DE Phase Algorithm

Input: Solution cat, Populations Cats, mutation factor F, crossover rate CR
Output: Updated Position of Solution cat

```
Select r1, r2, r3 randomly such that: r1 ≠ cat, r2 ≠ cat, r3 ≠ cat and r1 ≠ r2 ≠ r3
x1 ← Cats[r1], x2 ← Cats[r3], x3 ← Cats[r3]
For each dimension D Do  // mutation
    Mutant[D] ← x1.position[D] + F × (x2.position[D] – x3.position[D]   // Using (16)
End For
For each dimension D Do
    rand ← Uniform(0,1)
    If rand < CR Then
        trial[D] ← mutant[D]
    Else
        trial[D] ← cat.position[D]
    End If
End For
trialSol ← new Solution(trialSol)
Evaluate fitness(trialSol)
If fitness(trialSol) < fitness(cat) Then
    cat.position ← copy trial
End If
return cat
```

The next section describes the proposed hybrid algorithm in detail.

4.5 Proposed DWCSO Theory

The proposed algorithm is inspired by the behavior of Cat Swarm Optimization algorithm as discussed earlier. In the proposed cat swarm based algorithm DWCSO, we have injected Differential Evolution's mutation and crossover (Algorithm 2) for better exploration. The tracing mode of the proposed DWCSO is enhanced by SMIW inertia weight discussed in (14) to balance the exploration and exploitation phases i.e. seeking and tracing modes.

Algorithm 3: Hybrid DWCSO Algorithm

Input: Solution cat, Populations Cats, SMP, CDC, MR, max_Generations etc.
Output: Best Schedule gBest

```
For Gen = 0 to max_Generations Do
    Call Algorithm 2 for mutation and crossover  // Exploration
    Update Seeking and Tracing Modes using MR value
    For each cat in Cats Do
        If cat is in Seeking Mode Then
            run Seeking Mode     // Update positions, use (11)
        Else
            run Tracing Mode     // Update positions, use (13)
        End If
        If cat.fitness < gBest.fitness Then
            gBest ← copy (cat)
        End If
    End For
End For
return Best Schedule (gBest)
```

The working of the proposed algorithm is as follows:

- The proposed (Algorithm 3) DWCSO initializes the workflows or Google Cluster Traces 2019 using Algorithm 1 i.e. HEFT algorithm to maintain the parent-child dependency for scheduling of workflows. The dependency is considered 'nil' in case of independent workload i.e. Google Traces 2019.

- The DE phase applies for mutation and crossover to solutions for better exploration of solutions i.e. finds more solutions based on makespan and cost. This phase maintains population diversity for efficient results.
- The refined cats retrieved from DE phase i.e. refined solutions are distributed randomly in seeking and tracing modes based on the MR value 0.2 i.e. 80% cats to seeking mode whereas 20% in tracing mode for processing. The seeking mode is responsible for further exploration if any and tracing mode is executed for sharp exploitation i.e. hunting. The tracing mode achieves better exploitation due to the SMIW weight which also balances the seeking and tracing modes.
- After the 3rd step, the fitness of the solution is checked, if the solution is found better compared to previous solution, then the new solution is stored in memory.
- This process continues till the stopping criterion is met. Finally, the best solution is returned i.e. the optimized Tasks-VMs schedule.

The simulation parameters of the algorithms used in this research are represented in Table 4. The number of runs of each algorithm is 15 and the stopping criterion is maximum generations i.e. 200.

4.6 Novelty of Research

In our research, an adaptive inertia weight (SMIW) has been used for Tracing Mode of DWCSO algorithm to balance the Seeking and Tracing modes. The DE mutation phase has been injected for enhancing the exploration of the algorithm. Table 3 is presenting the detail regarding the gaps covered in this research.

Table 3: Research Novelty Comparison

Research	Workload Used	Parameters
DECSO [20]	Independent	Makespan, Migration Time
IGWO [28]	Workflows	Makespan, Cost, Energy
CSO [5]	Independent	Makespan, Energy, SLA Violation
Proposed Research	Both Workflows and Independent	Makespan, Cost and Throughput

The major parameters such as makespan, cost and throughput are used to compare the proposed DWCSO algorithm with other hybrid and state of art algorithms developed by the scientists.

5. Simulation Results and Discussion

The simulation is carried out using CloudSim 3.0 simulator written in Java programming language. We have implemented all algorithms given in the research papers for a fair analysis and executed the algorithms' code using NetBeans 8.0 on a machine having Processor 12th Gen Intel® Core™ i5-1235U, RAM 16 GB, Storage 512 GB and Windows 11 OS.

Table 4: Algorithms' Parameters

Proposed DWCSO Algorithm	
Parameter	**Value**
Max Generations	200
No. of Cats (Population)	50
SMP, SRD, CDC, MR, F, CR	5, 0.3, 0.3, 0.2, 0.8 and 0.9 respectively
c1, rand, ω_{max}, l, g	1.5, (0,1), 2.0, 2.0, and 2.0 respectively
CSO Algorithm Properties	
Parameter	**Value**
Max Generations	200
No. of Cats	50
SMP, MR, SRD, CDC, c1	5, 0.2, 0.3, 0.3 and 1.5 Respectively
IGWO Algorithm Properties	

Parameter	Value
Generations	200
No. of Wolves	50
F, CR, Flipping Probability	0.8, 0.9, 0.3

We have created 5 types of 60 heterogeneous VMs which are having computing power between 500-1000 MIPS, 512-1024 MB RAM and 1000 Mbps bandwidth. All the algorithms have been run for 15 rounds independently. The rest simulation parameters are described in Table 4. The performance parameters are makespan, cost and throughput as described earlier.

Further, this section is presenting the results obtained using two scenarios. We have conducted experiments for two scenarios as discussed in Table 5. For first scenario, we have used scientific workflows CyberShake, Montage, Inspiral and Sipht workflows dataset in XML format, each containing 1000 tasks. The structure of these workflows is available on https://pegasus.isi.edu. For second scenario, we have used Google Cluster 2019 Traces dataset; these are accessed through Kaggle repository in CSV format. The workload properties are also discussed in Table 5.

Table 5: Workload Properties

Scenario	Workload	No. of Tasks	Type	Length
1	CyberShake	1000	Dependent	Random as given in Dataset (All tasks are normalized by 500 MI multiplication)
	Montage			
	Inspiral			
	Sipht			
2	Google Traces	500	Independent	Random 5000-416851 Based on CPU and Run Time Requirements (Normalized missing tasks by 5000-15000 MI and rest tasks with 1000 MI multiplication)
	Google Traces	1000		
	Google Traces	1500		
	Google Traces	2000		

5.1 First Scenario Results and Discussion

In first scenario, four most widely used scientific workloads CyberShake, Montage, Inspiral, and Sipht have been scheduled on 60 VMs. The experiments were simulated using proposed DWCSO, IGWO [28], PSO and CSO algorithms and the mean results of 15 runs are compared in Fig. 3. The evaluation of all algorithms was carried out using three important cloud scheduling metrics makespan, cost and throughput. In all parts of the figure, the x-axis is showing algorithms and the y-axis is representing metrics values. Lower makespan, lower cost and higher throughput are considered better. The experiments results clearly indicate that the proposed DWCSO algorithm consistently outperforms the competing algorithms across all workloads.

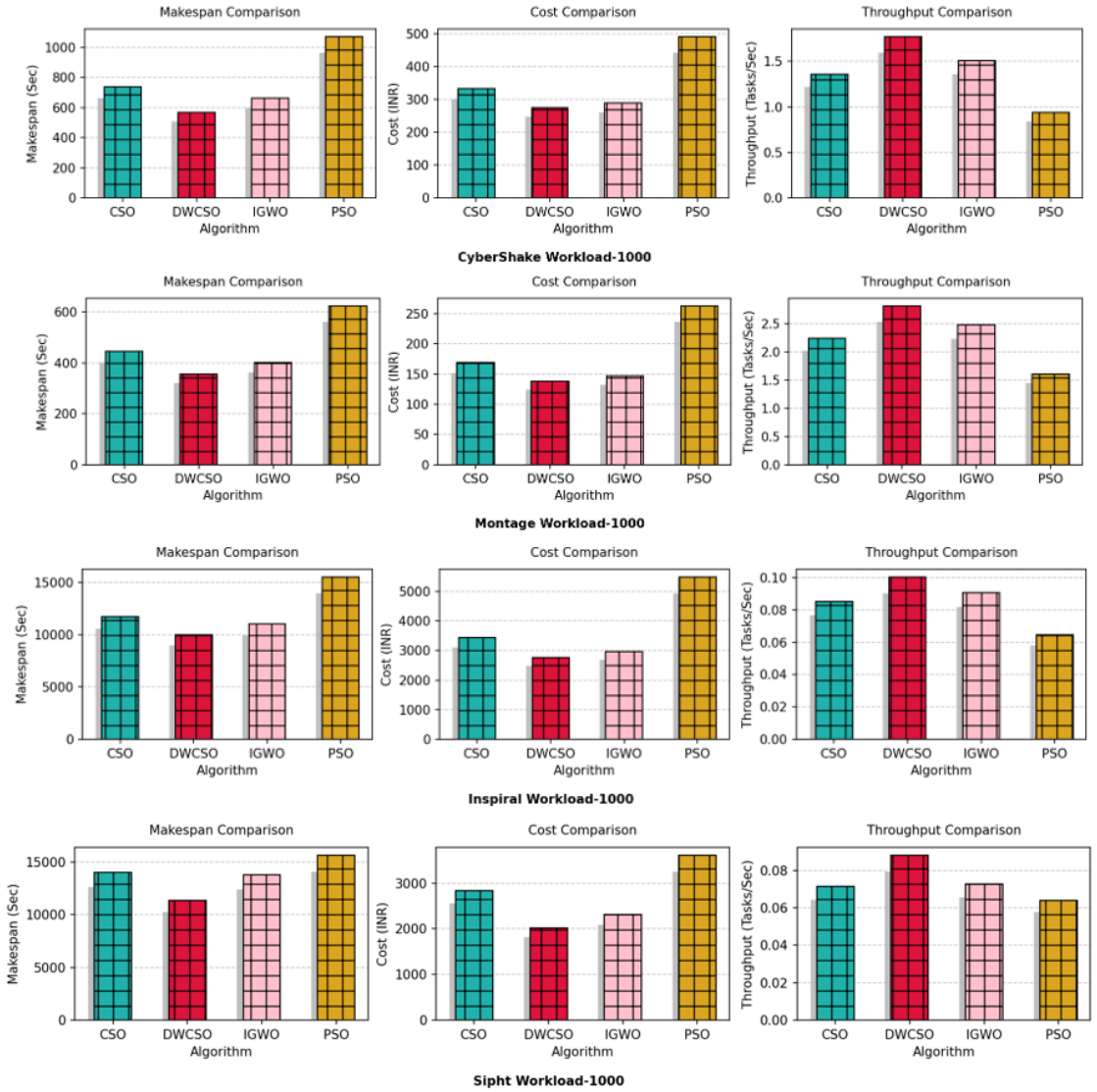

Fig. 3: CyberShake, Montage, Inspiral and Sipht Workloads Scheduling Results

For the CyberShake workload, DWCSO achieved the minimum makespan compared to IGWO [28], CSO and PSO, indicating its ability to complete workflow execution in significantly less time. A lower makespan indicates improved scheduling efficiency and better utilization of available virtual machines in one or more host machines. Whereas PSO shown the higher makespan due to inefficient resource allocation and slower convergence behavior. Likewise, the execution cost generated by proposed DWCSO was lowest among all algorithms, while PSO utilized the maximum cost. The reduction in cost recognized to the optimized task-to-VM mapping strategy adopted by DWCSO, which minimizes unnecessary VM idle time. Furthermore, DWCSO achieved highest throughput, proving its capability to process a larger number of tasks per second.

A similar performance trend was found for Montage workload. The proposed DWCSO again produced the lowest makespan and execution cost while leading the highest throughput. The results clearly indicate that the proposed algorithm effectively balances the workload among virtual machines and avoids maximum tasks overloading. Although IGWO [28] performed better than CSO and PSO, it is still weaker than DWCSO in all performance metrics. PSO consistently showed poor performance due to premature convergence and limited exploration power during the optimization process.

For the Inspiral workload a computationally rigorous and more complex dataset than the previous said workflows, the proposed DWCSO maintained superior performance by significantly reducing makespan and operational cost. The makespan achieved by DWCSO was noticeably lower than PSO algorithm. It is demonstrating the scalability and robustness of the proposed scheduling algorithm under heavy workload conditions. In addition, the throughput obtained by DWCSO was highest among all considered algorithms. It is indicating efficient task execution and improved resource utilization even for large-scale workflows.

The Sipht workload also proved the effectiveness of the proposed algorithm DWCSO. It consistently outperformed CSO, IGWO [28] and PSO by minimizing execution time and cost while maximizing throughput. The observed improvements confirm that the dynamic scheduling mechanism and adaptive optimization capability of DWCSO ensure better exploration and exploitation during the entire scheduling process. Thus, the algorithm achieves balanced VM load distribution and faster convergence near optimal scheduling solutions.

Overall, the experimental analysis presented that the proposed DWCSO algorithm provides an efficient and reliable scheduling due to its enhanced search capability, adaptive weight adjustment method and effective balance between seeking and tracing modes.

5.1.1 First Scenario Convergence Analysis

The convergence curves displayed in Fig. 4 (a-d) demonstrate the proposed DWCSO, IGWO [28], CSO and PSO algorithms for different scientific workflows including CyberShake, Montage, Inspiral and Sipht with 1000 tasks. The x-axis represents the number of generations, whereas the y-axis shows the best fitness value of the fitness function obtained during optimization process. In this research, the objective of workload scheduling is to minimize the fitness value; a lower curve indicates better optimization performance and faster convergence toward the near optimal solution.

It can be observed that the proposed DWCSO algorithm converges faster than CSO, IGWO [28], and PSO across all workloads. During the initial generations, all algorithms show a rapid decrease in fitness values due the exploration process. However, DWCSO reaches a stable minimum fitness value much earlier that is demonstrating its strong exploitation capability and efficient search mechanism. In opposition, PSO shows slower convergence and stabilizes at higher fitness values. It indicates a premature convergence and reduced optimization efficiency. Similarly, CSO and IGWO [28] achieve moderate convergence performance but fail to attain fitness values as low as produced by the proposed DWCSO algorithm.

The convergence analysis proves that the proposed DWCSO algorithm has superior optimization efficiency, faster convergence speed, and powerful global search capability compared to CSO, IGWO, and PSO algorithms. The improved convergence is due to the adaptive search strategy and better balance between exploration and exploitation injected in the proposed DWCSO algorithm. Therefore, the algorithm is able to generate high-quality scheduling solutions with lower computational effort. It makes it highly suitable for complex cloud workflow scheduling environments.

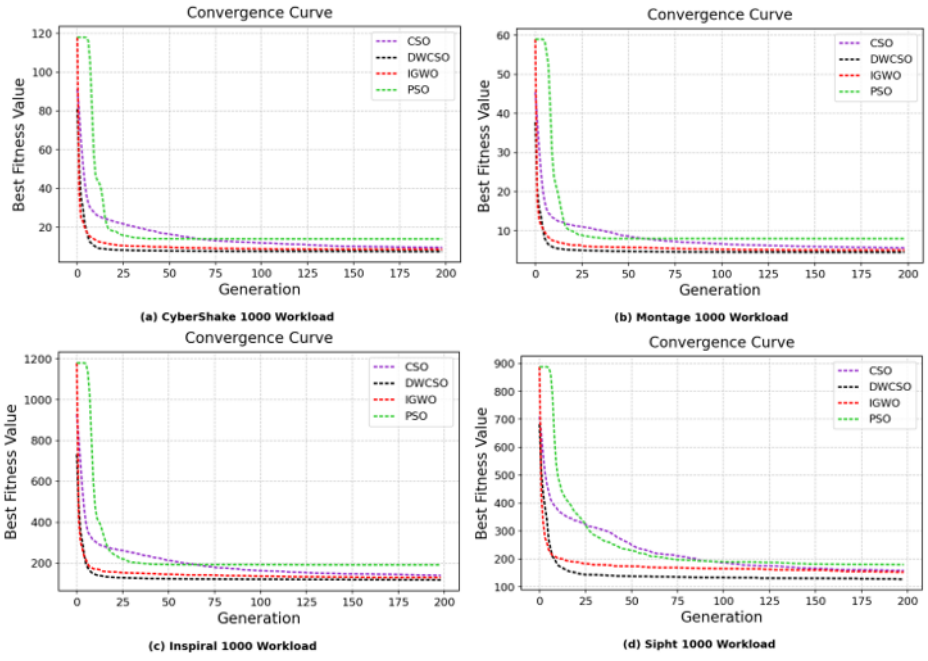

Fig. 4: Algorithms Convergence for CyberShake, Montage, Inspiral and Sipht Workloads

5.2 Second Scenario Results and Discussion

In this scenario, we have considered independent tasks scheduling. We have implemented and run the proposed DWCSO, IGWO [28], PSO, and CSO algorithms for real world dataset Google Cluster 2019 traces. Fig. 5 (a-c) presents the comparative performance analysis of all algorithms under varying number of tasks

ranging from 500 to 2000 independent tasks in order to evaluate the scalability and efficiency of the algorithms. The performance was analyzed using three important metrics: makespan, execution cost and throughput.

In this scenario, we have considered independent tasks scheduling. We have implemented and run the proposed DWCSO, IGWO [28], PSO, and CSO algorithms for real world dataset Google Cluster 2019 traces. Fig. 5 (a-c) presents the comparative performance analysis of all algorithms under varying number of tasks ranging from 500 to 2000 independent tasks in order to evaluate the scalability and efficiency of the algorithms. The performance was analyzed using three important metrics: makespan, execution cost and throughput.

Fig. 5 (a) illustrates the variation of makespan with respect to the number of tasks. It can be observed that the makespan increases for all algorithms as the workload size grows due to the higher computational demand and resource utilization. Still, the proposed DWCSO algorithm consistently achieves the lower makespan across all task sizes. For 500 tasks, the DWCSO keeps lowest makespan compared to CSO, IGWO, and PSO. The performance advantage becomes more significant as the number of tasks increases. At 2000 tasks, PSO consumes the highest makespan and giving poor scalability and inefficient resource scheduling under large workloads. In disparity, the proposed DWCSO demonstrates better workload balancing and optimized task allocation which significantly reduces tasks finishing time i.e. makespan. IGWO algorithm performs better than PSO and CSO but remains weaker than DWCSO.

Fig. 5 (b) presents the execution cost analysis for different task sizes. Similar to the makespan, the execution cost increases with the increase in workload size because additional cloud resources and VM execution time are required. However, DWCSO consistently produces the minimum execution cost among all compared algorithms. The lower cost achieved by DWCSO indicates efficient utilization of virtual machine resources and reduces the overloaded conditions during task scheduling. PSO earns the maximum cost particularly for larger workloads due to inefficient convergence and poor optimization capability. Although IGWO shows moderate improvement over CSO and PSO but it still does not match the cost efficiency achieved by DWCSO.

Fig. 5 (c) depicts the throughput comparison of all scheduling algorithms. Throughput represents the number of tasks completed per second; hence, higher throughput indicates better scheduling performance and resource utilization. The results indicate that the DWCSO consistently achieves the highest throughput for all workload sizes. Its throughput slightly increases as the number of tasks grows. This behavior demonstrates the scalability and robustness of the proposed algorithm DWCSO in handling large-scale cloud workloads. In opposite to it, PSO exhibits the lowest throughput which gradually decreases with increasing task size due to inefficient scheduling decisions and slower convergence. CSO and IGWO achieve moderate throughput performance but remain lower to DWCSO.

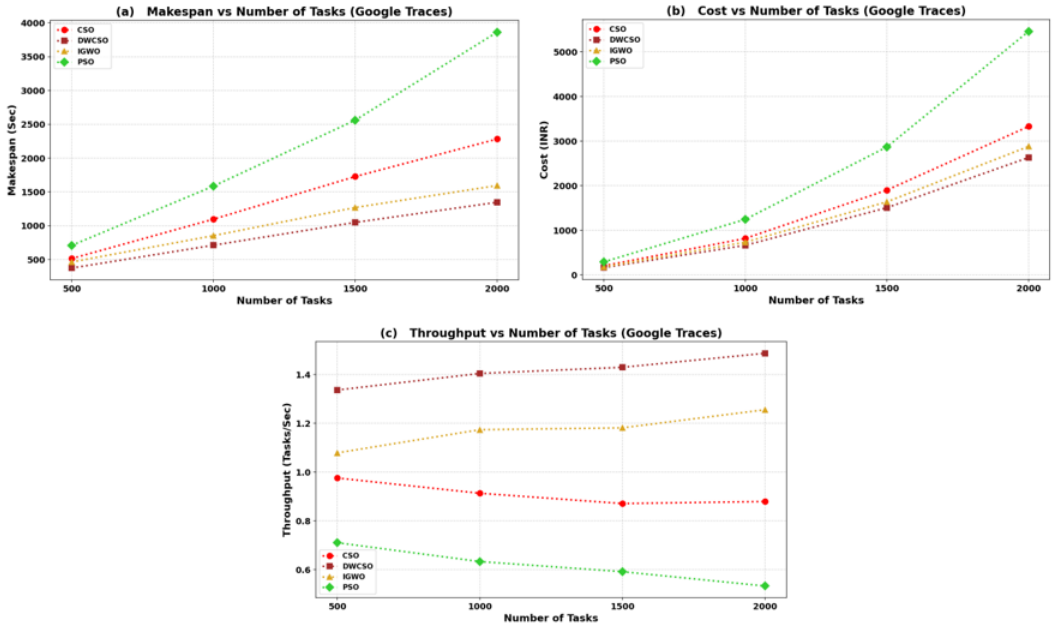

Fig. 5: Makespan, Cost and Throughput Results Using Google Traces 2019

The results analysis proves that the proposed DWCSO algorithm provides higher scalability and scheduling efficiency compared to CSO, IGWO, and PSO under varying workload conditions. The improved performance of DWCSO is due to its adaptive optimization strategy, balanced exploration-exploitation mechanism, and effective task-to-resource mapping capability. The proposed DWCSO algorithm proves itself a reliable and efficient algorithm for large-scale task scheduling in cloud computing.

5.2.1 Second Scenario Convergence Analysis

Fig. 6 (a-d) presents the convergence analysis of the proposed DWCSO, CSO, PSO and IGWO [28] algorithms for Google Traces workloads with different takes sizes ranging from 500 to 2000 tasks. The x-axis represents the number of generations while the y-axis depicts the best fitness value of the fitness function achieved during optimization process. The objective of task scheduling is to minimize the fitness value. So, lower convergence curves indicate better optimization performance and higher scheduling effectiveness. The convergence plots evident that all algorithms experience a rapid reduction in fitness values during the initial generations due to the exploration phase. However, the proposed DWCSO algorithm consistently demonstrates faster convergence and achieves lower final fitness values compared to CSO, IGWO, and PSO algorithms. This type of behavior indicates that the proposed DWCSO algorithm has a stronger capability to identify near-optimal solution within fewer generations.

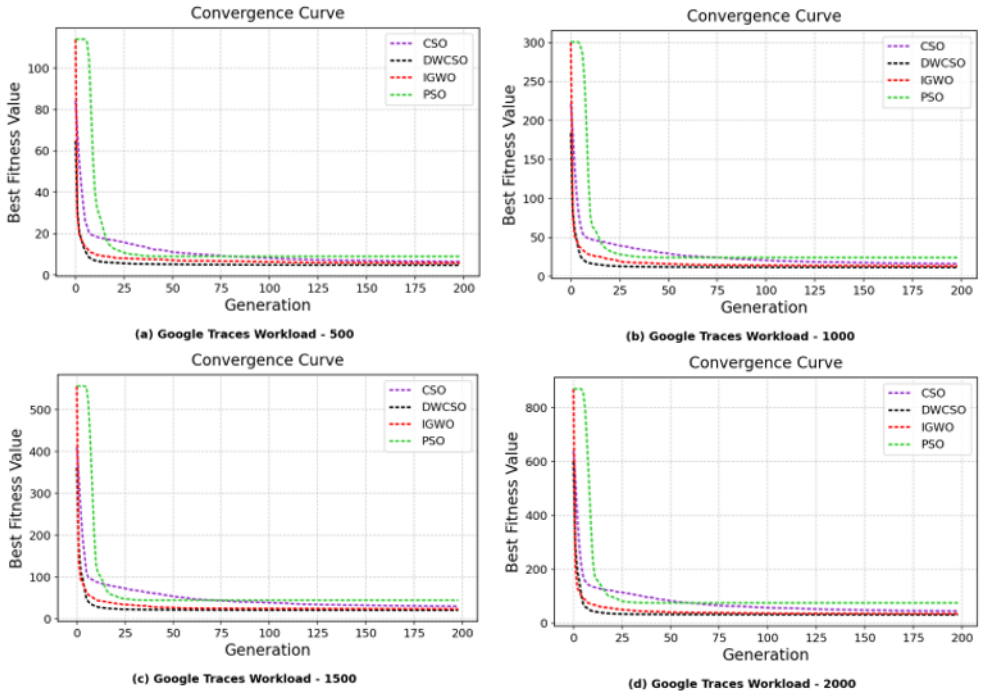

Fig. 6: Algorithms Convergence Results for Google Traces 2019 Workloads

Fig. 6 (a) represents the Google trace workload with 500 tasks. DWCSO rapidly decreases the fitness value and stabilizes earlier than the other algorithms.

The faster stabilization of the curve indicates efficient exploration and reduced computational efforts. Although CSO and IGWO also show gradual improvements but their convergence speed is slower than DWCSO. They also settle at higher fitness values. The PSO algorithm demonstrates the poorest performance and it requires more generations to convergence and producing the highest final fitness value due to premature convergence.

A similar trend is observed in Fig. 6 (b) for the workload containing 1000 tasks. The proposed DWCSO achieves the minimum fitness value of the fitness function (makespan and cost) within a small relatively small number of generations, demonstrating its ability to efficiently balance exploration and exploitation during the search process. In contrast, IGWO and CSO converge more slowly and fail to reach the lowest fitness values achieved by DWCSO. PSO again shows weak optimization capability and stabilizes at a significantly higher fitness level.

For the larger workload of 1500 tasks shown in Fig. 6 (c), the superiority of DWCSO becomes more pronounced. As the workload increases the proposed algorithm DWCSO maintains stable and rapid

convergence behavior. The competing algorithms require more iteration to get converged. The reduced fitness values achieved by DWCSO indicate improved scheduling quality. It proves efficient task allocation and better resource utilization in a large-scale cloud environment.

Fig. 6 (d) represents the workload with 2000 tasks and demonstrates that DWCSO continues to outperform other algorithms under heavy computational workloads. The convergence curve of DWCSO reaches stability much easier and maintains the lowest fitness value throughout the optimization process.

Table 6: Results Summary for Workflows

Workload	Metric	Result Type	IGWO	DWCSO	PSO	CSO
CyberShake	Makespan	Mean	662.25	**565.4**	1070.02	736.55
		Best	611.2	**539.0**	935.25	661.05
		Worst	723.99	**614.61**	1258.26	805.73
	Cost	Mean	289.48	**275.2**	490.59	333.51
		Best	275.66	**254.34**	432.1	304.19
		Worst	307.88	**300.09**	577.3	377.97
Montage	Makespan	Mean	402.77	**355.79**	622.46	445.76
		Best	378.1	**342.02**	525.2	425.39
		Worst	423.85	**375.25**	687.64	461.44
	Cost	Mean	147.12	**138.23**	262.02	168.61
		Best	141.78	**129.01**	207.54	160.35
		Worst	153.59	**150.45**	308.08	187.27
Inspiral	Makespan	Mean	11012.26	**9975.72**	15486.13	11740.69
		Best	10507.02	**9426.27**	13396.38	11181.83
		Worst	11634.77	**10391.88**	17229.74	12436.53
	Cost	Mean	2974.55	**2762.27**	5477.25	3442.04
		Best	2815.95	**2585.34**	4711.22	3126.08
		Worst	3210.62	**2914.89**	6482.21	3967.08
Sipht	Makespan	Mean	13746.42	**11363.84**	15594.53	14014.74
		Best	12224.3	**10599.34**	12878.83	12059.79
		Worst	14699.52	**12001.79**	18112.83	15465.54
	Cost	Mean	2313.74	**2023.23**	3606.62	2841.87
		Best	2063.65	**1703.77**	3039.51	2431.55
		Worst	2705.47	**2309.84**	4208.59	3896.21

Table 7: Results Summary for Google Traces 2019

Workload	Metric	Result Type	IGWO	DWCSO	PSO	CSO
Google Traces 500	Makespan	Mean	464.41	**374.69**	709.54	515.01
		Best	410.47	**350.6**	585.95	465.92
		Worst	499.34	**414.37**	805.87	596.33
	Cost	Mean	181.18	**162.71**	291.74	204.78
		Best	170.31	**155.28**	250.47	184.32

		Worst	203.56	**172.43**	326.44	236.52
Google Traces 1000	Makespan	Mean	853.91	**712.5**	1586.32	1097.64
		Best	747.1	**672.93**	1405.81	975.55
		Worst	906.32	**749.38**	1770.25	1197.2
	Cost	Mean	733.88	**663.65**	1245.44	818.01
		Best	684.28	**619.64**	1048.85	754.83
		Worst	766.32	**712.98**	1369.85	921.01
Google Traces 1500	Makespan	Mean	1270.13	**1050.0**	2562.05	1728.03
		Best	1221.36	**1000.98**	2170.0	1495.39
		Worst	1338.08	**1114.62**	3129.06	1850.87
	Cost	Mean	1638.82	**1506.83**	2869.78	1901.34
		Best	1524.33	**1451.99**	2471.11	1711.32
		Worst	1716.9	**1557.04**	3380.57	2088.93
Google Traces 2000	Makespan	Mean	1594.96	**1347.38**	3863.48	2282.68
		Best	1512.8	**1230.83**	3077.32	2079.65
		Worst	1704.48	**1474.43**	5253.06	2649.39
	Cost	Mean	2879.93	**2631.22**	5459.54	3328.29
		Best	2795.49	**2547.12**	4692.49	3186.09
		Worst	3056.17	**2800.94**	6543.49	3575.26

PSO again shows slowest convergence and highest fitness values which indicates poor scalability and reduced optimization efficiency. The CSO and IGWO achieve moderate performance but remain weak than DWCSO in both convergence speed and solution quality.

The convergence analysis confirms that the proposed DWCSO algorithm demonstrates superior optimization capability. It shows faster convergence speed and improved stability compared to CSO, IGWO [28] and PSO for Google trace workloads of different sizes. The enhanced convergence behavior of DWCSO is due to its adaptive search mechanism and effective exploration-exploitation balance. These characteristics of DWCSO algorithm produce high-quality scheduling solutions with lower computational overhead and making it highly suitable for large-scale and dynamic cloud computing environments.

The results of both scenarios related to makespan and costs are also displayed in Table 6 and 7 for better clarity. The experimental results demonstrate that the proposed DWCSO algorithm consistently outperforms IGWO, CSO and PSO algorithms across all dependent and independent workloads in terms of makespan and cost. The improvement is more significant for larger and complex workloads. It is indicating the scalability and robustness of the proposed algorithm. On workflow applications, the proposed DWCSO achieves an average improvement of 10.46% over IGWO and 20.18% over CSO. Similarly, for Google Traces 2019 workloads, DWCSO provides average improvements of 13.15% over IGWO and 27.96% over CSO. These results confirm that the hybrid search and adaptive scheduling strategy of DWCSO effectively balances exploration and exploitation, leading to better resource utilization and reduced execution overhead.

6. Conclusion and Future Scope

This paper presented a novel hybrid DWCSO task scheduling algorithm for cloud computing by integrating Differential Evolution operators and an adaptive hybrid inertia weight mechanism (SMIW) to effectively balance exploration and exploitation. The proposed algorithm has been tested on various scientific workflows and Google traces 2019 independent tasks. Extensive experiments conducted on scientific workflows and Google Cluster 2019 Traces independent workloads demonstrate that DWCSO consistently outperforms other algorithms IGWO, CSO and PSO. After evaluation, it is observed that the proposed DWCSO algorithm consistently outperforms its competitor IGWO, achieving 9.41 to 17.33% improvement in makespan, 4.93 to 12.56% in cost with higher throughput in case of workflows. In case of Google Traces 2019, the proposed DWCSO achieves 15.52 to 19.32% improvement in makespan, 8.05 to 10.20% in cost over IGWO with higher throughput. The maximum improvement observed against PSO demonstrates the

superior efficiency of the proposed algorithm.

Future work can extend the proposed DWCSO algorithm by evaluating it on another real-world datasets. The algorithm can also be implemented in more complex cloud environments involving multiple hosts and data-centers. In addition, other machine learning techniques can also be incorporated for further enhancement of the scheduling process.

References

1. M. Y. Mulge, "Optimization of Task Scheduling Algorithm Using Modified Mean Grey-Wolf," International Journal of Intelligent Engineering and Systems, vol. 12, no. 4, pp. 192–200, 2019.
2. N. Arora and R. K. Banyal, "A Particle Grey Wolf Hybrid Algorithm for Workflow Scheduling in Cloud Computing," Wireless Personal Communications, vol. 122, pp. 3313–3345, 2021.
3. Z. Benlalia, K. Abouelmehdi, A. Beni-Hssane, and H. Khaloufi, "New Hybrid Algorithm for Task Scheduling in Cloud Computing," Journal of Theoretical and Applied Information Technology, vol. 99, no. 24, pp. 6272–6280, 2021.
4. P. Krishnadoss, C. Chandrashekar, and V. K. Poornachary, "RCOA Scheduler: Rider Cuckoo Optimization Algorithm for Task Scheduling in Cloud Computing," International Journal of Intelligent Engineering and Systems, vol. 15, no. 5, pp. 505–515, 2022.
5. S. Mangalampalli, S. K. Swain, T. Chakrabarti, P. Chakrabarti, G. R. Kerri, M. Margala, B. Unhelkar, and S. B. Krishnan, "Prioritized Task-Scheduling Algorithm in Cloud Computing Using Cat Swarm Optimization," Sensors, vol. 23, no. 13, pp. 6155, 2023.
6. S. H. Anbarkhan and M. A. Rakrouki, "An Enhanced PSO Algorithm for Scheduling Workflow Tasks in Cloud Computing," Electronics, vol. 12, no. 12, pp. 2580, 2023.
7. K. A. Tabary, H. Motameni, B. Barzegar, E. Akbari, H. Shirgahi, and M. Mokhtari, "QoS Aware Task Scheduling Using Hybrid Genetic Algorithm in Cloud Computing," IEEE Access, vol. 12, pp. 1–15, 2024.
8. M. Selselejoo and H. R. Ahmadifar, "DT-GWO: A Hybrid Decision Tree and GWO-Based Algorithm for Task Scheduling," Applied Soft Computing, 2024.
9. Z. A. Khan, I. A. Aziz, N. A. Osman, and S. Nabi, "Parallel Enhanced Whale Optimization Algorithm for Independent Tasks Scheduling on Cloud Computing," IEEE Access, vol. 12, pp. 23529–23545, 2024.
10. I. Behera and S. Sobhanayak, "Task Scheduling Optimization in Heterogeneous Cloud Computing Environments: A Hybrid GA-GWO Approach," Journal of Parallel and Distributed Computing, vol. 183, pp. 104766, 2024.
11. H. Mikram and S. El Kafhali, "CHPSO: An Efficient Algorithm for Task Scheduling and Resource Optimization in Cloud Environment," Journal of Grid Computing, vol. 23, pp. 15, 2025.
12. S. Khurana and R. K. Singh, "Workflow Scheduling and reliability improvement by hybrid intelligence optimization approach with task ranking," EAI Endorsed Transactions on Scalable Information Systems, vol. 7, no. 24, pp. 1-10, 2019.
13. D. Ramesh, S. S. Kolla, D. Naik, and R. Narvaneni, "HGWO-MultiQoS: A Hybrid Grey Wolf Optimization Approach for Workflow Scheduling," Simulation Modelling Practice and Theory, vol. 142, pp. 103127, 2025.
14. S. Abdi, M. Ashjaei, and S. Mubeen, "Deadline-Constrained Security-Aware Workflow Scheduling in Hybrid Cloud Architecture," Future Generation Computer Systems, vol. 162, pp. 107466, 2025.
15. A. Saeed, G. Chen, H. Ma, and Q. Fu, "A Genetic Algorithm with Selective Repair for Deadline-Constrained IoT Workflow Scheduling," Future Generation Computer Systems, 2025.
16. L. Abualigah, A. M. A. Hussein, M. H. Almomani, R. A. Zitar, H. Migdady, A. I. Alzahrani, and A. Alwadain, "Improved Synergistic Swarm Optimization Algorithm for Task Scheduling in Cloud Computing," 2024.
17. J. Bhagwan, and S. Kumar, "An H-CSO Algorithm for Workflows Scheduling in Heterogeneous Cloud Environment," International Journal of Intelligent Engineering & Systems, vol. 14, no. 5, pp. 422-434, 2021.
18. S. Mangalampalli, S. K. Swain, and V.K. Mangalampalli, "Multi-Objective Task Scheduling in Cloud Computing Using Cat Swarm Optimization Algorithm," Arabian Journal of Science & Engineering," vol. 47, pp. 1821-1830, 2022.
19. J. Bhagwan, and S. Kumar, "An HC-CSO Algorithm for Workflow Scheduling in Heterogeneous Cloud Computing System," IJACSA, vol. 12, no. 6, pp. 484–492, 2021.
20. A. Gunduz, S. Senan, and Z. Gurkas-Aydin, "A Hybrid DECSO Algorithm for Efficient Multi-Objective Task Scheduling," Cluster Computing, vol. 28, pp. 848, 2025.
21. S. Chu, P. Tsai and J. Pan, "Cat Swam Optimization," In: Proc. of LNAI, vol. 4099, pp. 854-858, 2006.
22. P. Bansal, S. Kumar, S. Pasrija, and S. Singh, "A Hybrid Grasshopper and New Cat Swarm Optimization Algorithm for Feature Selection and Optimization of Multi-Layer Perceptron," Soft Computing, vol. 24, pp. 15463-15489, 2020.
23. S. Chen, Z. Xu, Y. Tang, and S. Liu, "An Improved Particle Swarm Optimization Algorithm Based on Centroid and Exponential Inertia Weight," Mathematical Problems in Engineering, vol. 2014, pp. 1-14, 2014.
24. T. O. Ting, Y. Shi, S. Cheng, and S. Lee, "Exponential Inertia Weight for Particle Swarm Optimization," In: Proc. of Lecture Notes in Computer Science, pp. 83-90, 2012.
25. K. Li, L. Jia, and X. Shi, "Research on Cloud Computing Task Scheduling Based on PSOMC," Journal of Web Engineering, vol. 21, no. 6, pp. 1749-1766, 2022.
26. H. Topcuoglu, H. Hariri, and M. Y. Wu, "Performance Effective and Low Complexity Task Scheduling for

Heterogeneous Computing," IEEE Transactions on Parallel and Distributed Systems, vol. 13, no. 3, pp. 260-274 2002.
27. F. A. Hasim, E. H. Houssein, K. Hussain, M. S. Mabrouk, and W. Al-Atabany, "Honey Badger Algorithm: New Metaheuristic Algorithm for solving optimization problems," Mathematics and Computers in Simulation, vol. 192, pp. 84-110, 2022.
28. M. M. Nezami, and A. Kumar, "Energy Aware Workflow Allocation in Cloud Computing Using Improved Grey Wolf Optimization," International Journal of Advanced Computer Science & Applications, vol. 16, no. 10, pp. 492-498, 2025.
29. Y. Zhao, "Enhanced Butterfly Optimization Algorithm for Task Scheduling in Cloud Computing Environments," International Journal of Advanced Computer Science and Applications, vol. 15, no. 12, pp. 435-443, 2024.
30. J. Bhagwan, S. Rani, Manoj, S. Godara, Yashasvi, and S. Kumar, "IJPSO: An Improved Hybrid PSO Algorithm for Multi-Type and Large Scale Data Scheduling in Cloud Computing," International Journal of Artificial Intelligence and Machine Learning, vol. 6, no. 3, pp. 399-412, 2026.