SvedbergOpen
DISSEMINATION OF KNOWLEDGE

International Journal of Artificial Intelligence and Machine Learning

Publisher's Home Page: https://www.svedbergopen.com/

Research Paper Open Access

HICSO Algorithm for Performance and Cost Optimization Based Workflow Scheduling in Cloud Computing

Jai Bhagwan[1*], Sandeep[2], Ajay Kumar[3], Partibha Yadav[3], Palak[4], Mamta Yadav[5], Parshant Bansal[3], Jogender Singh Yadav[3]

[1]Department of Computer Science & Engineering, Guru Jambheshwar University of Science & Technology, Hisar, India
[2]Department of Artificial Intelligence & Data Science, Guru Jambheshwar University of Science & Technology, Hisar, India
[3]Department of Computer Science & Engineering, Indira Gandhi University, Meerpur, Haryana, India
[4]Department of Computer Science, Govt. College, Narnaul, Haryana, India
[5]Department of Computer Science and Information Technology, Central University of Haryana, Mahendergarh, India
*Corresponding Author: drjaicse@gmail.com

Abstract
Cloud workflow scheduling in heterogeneous environments is a challenging optimization problem due to the large task–VM search space and the need to achieve efficient resource utilization. This study proposes HICSO (Hybrid Inertia-weight-based Cat Swarm Optimization), an improved Cat Swarm Optimization algorithm incorporating a proposed Hybrid Inertia Weight (HIW) strategy to improve the balance between exploration and exploitation. HEFT is used to generate the initial population, while HIW is integrated into the tracing mode of CSO. The proposed approach jointly optimizes makespan and cost using a weighted fitness function. Experiments are conducted in CloudSim 3.0 using four scientific workflows—CyberShake, Montage, Inspiral, and Sipht—with 1000 tasks each and 70 heterogeneous VMs. The algorithms are executed for 15 independent runs for comparative evaluation. Results show that HICSO achieves the lowest mean makespan and mean cost across all four workloads, demonstrating improved scheduling efficiency and cost effectiveness compared with LDCSO, CICSO, OICSO, and conventional CSO.

1 Introduction

Cloud computing has become a crucial platform for running computationally demanding applications due to its capacity of offering diverse computing resources. The growing quantity of tasks and heterogeneous virtual machines render task scheduling as a difficult optimization problem. A task scheduler must establish a suitable connection between tasks and VMs while minimizing makespan, cost, energy and increase throughput. With the increment of possible combinations of tasks and resource i.e. problem scale, traditional scheduling methods fall short in delivering high-quality solutions. This has attracted the adoption of population centred swarm intelligence algorithm. Particle Swarm Optimization and Cat Swarm Optimization methods gained a significant focus in scheduling in recent years. PSO was first introduced by Kennedy and Eberhart which is based on the social behaviour of birds and fish within a population [1]. This approach paints possible solutions as particles navigating through the search space based on their personal and social experiences. Shi and Eberhart incorporated the inertia weight into PSO to manage effect of prior velocity on present motion [2]. Higher inertia weight enables particles to move more broadly through the search space and enhance exploration while the lower value allows particles to focus on more promising areas to enhance exploitations. So, the inertia weight is crucial in influencing the convergence of PSO. A constant inertia weight may not offer suitable equilibrium between exploration and exploitation during the optimization process. In the initial stages of scheduling the exploration requires satisfactory variety of solutions i.e. various task-VMs allocations. As the optimization reaches to advance stages, the searching should be increasingly narrow to allow refinement in promising schedules. This concept has inspired scientist to create various types of inertia weight strategies such as adaptive, linearly diminishing, time-varying, nonlinear and problem-specific inertia weight methods. In cloud scheduling these methods plays significant role due to discrete, diverse and heavily limited nature of the solution space. For example, Zhao et al. enhanced the adaptive inertia-weight method by considering the particle's fitness as well as the global and local optimal positions [3]. Wang et al. proposed an enhanced PSO that used adaptive inertia weight with random factor correlation approach which caused reduction in cost [4]. Zhou et al. created an enhanced PSO for scheduling cloud tasks where the inertia weight was modified dynamically to enhance the scheduling results [5]. Huang et al. explored various explored various time varying inertia weight

strategies for PSO based cloud scheduling [6]. These investigations show that inertia weight selection is not just setting the parameters as it can significantly influence the quality of the convergence of cloud scheduling methods.

CSO offers an alternative swarm oriented search method which is introduced by Chu, Tsai and Pan. This method has two distinct behaviours of cats: seeking state and tracing state [7]. Seeking state facilitates the exploration analysis and tracing state allows progress toward appealing solutions i.e. exploitation. This blend renders CSO appealing for scheduling issues where both local and global optimizations are needed. Bilgaiyan et al. used CSO for scheduling of workflows in cloud computing execution and data transmission costs [8]. The finding proved that CSO could achieve competitive tasks-VMs mappings and surpass PSO regarding load distribution. A multi-objective extension was later developed to take up multiple scheduling issues at the same time [9]. Subsequent research integrated inertia weight methods into CSO which shows that managing the movement behaviour can enhance the scheduling effectiveness in cloud [10]. The scheduling based on CSO has also been expanded to consider task priorities, SLA breaches, energy usages and real-time workflows [11]. So, both algorithms illustrate a significant role in managing the exploration and exploration balance. A thorough examination of inertia weight techniques in CSO based algorithms can offer a valuable contribution in cloud tasks scheduling approaches.

The main contributions of this research paper are:

1. Proposed Hybrid Inertia Weight to offer workflows scheduling in cloud computing.
2. Designed an Improved Cat Swarm Optimization named HICSO with the help of Hybrid Inertia Weight.
3. Considered Makespan and Cost optimization for workflows scheduling.
4. Comparison of proposed HICSO algorithm with other CSO family algorithms.

In rest paper, literature can be read in section 2, section 3 represents problem's formulation, data and methods are represented in section 4, simulation setting and results are discussed in section 5. Finally, conclusions are drawn in section 5.

2 Literature Review

Studies on swarm-based cloud task scheduling have gradually shifted from tradition to adaptive methods that can adjust to evolving search environments. Shi and Eberhart implemented the inertia weight to regulate the particle speed and enhance the exploration and exploitation balance [2]. This advancement played a crucial role for scheduling as too much exploration can hinder the convergence while too much exploitation may lead the optimizer to get stuck locally during task-VMs assignment. The significance of adaptive inertia control has been identified in cloud task management. Zhao et al. suggested an enhanced PSO where inertia weight was modified based on particles fitness along with personal and global best positions [3]. Their target was to define particle's search condition with higher precision and restrict early convergence. Wang et al proposed an enhance PSO that used adaptive inertia weight strategy with integration of random factor correlation mechanism [4]. Their experiments demonstrated reduced scheduling costs compared to other existing methods. The suggestion was that an inertia weight adjustment can affect real world scheduling efficiency. Zhou et al. designed a revised PSO tailored for task scheduling in cloud. They recognized that inertia weight is crucial element of scheduling method [5]. Their study tackled the challenge of creating effective scheduling in cloud and employed a dynamic modification to enhance the search process. Huang et al. compared time dependent inertia weight strategies for cloud scheduling [6]. Their research explored multiple distinct PSO variants and produced diverse convergence traits and scheduling results.

The literature on CSO has somewhat a distinct evolution. Bilgaiyan et al. utilized CSO for workflow scheduling in cloud computing. Both execution and data transmission expenses were considered [8]. Their simulation experiments demonstrated that CSO can achieve better results in fewer iteration compared to PSO algorithm. The same research team later designed a multi-objective CSO method for cloud workflow scheduling and presented the algorithm for various scheduling needs [9]. Gabi et al. highlighted the significance of search control in CSO by introducing a cloud scheduling technique based on CSO. This research integrated LDIW inertia method into tracing state [10]. The goal of the research was to improve the weakness of CSO and enhance makespan and convergence results. This research is actually relevant to the current study as it shows that the idea of inertia weight control initially used in PSO can be integrated in CSO for enhancing convergence and searching. Gabi et al. introduced a chaotic CSO method for cloud tasks scheduling in order to tackle premature convergence load imbalance [11]. The described research focused on scheduling challenges related to execution duration and expenses demonstrating that how limitations of CSO can be reduced. A novel scheduling method was introduced by deciding task priorities and accessing makespan, energy use and SLA breaches [12].

Collectively these studies make inspiration to introduce a hybrid inertia weight for CSO algorithm to address the scheduling issues.

3 Problem's Formulation

This research is addressing the challenge of real world workflows scheduling in heterogeneous environment of cloud computing. In this research, 70 heterogeneous virtual machines have been created in one host and one datacenter. The virtual machines characteristics are displayed in Table 2.

Table 2: VMs Characteristics

VM Parameters	Values
Computing Power	500-1000 MIPS
RAM	512-1024
Bandwidth	1000 bps
Types	5 (low to high capacity)

Further, this research allocates scientific workflows on heterogeneous VMs. Each workflow mentioned ahead is having 1000 tasks.

Let $T = \{T_1, T_2, T_3, T_4, T_5, \dots, T_n\}$ be the set of n tasks and $VM = \{VM_1, VM_2, VM_3, VM_4, VM_5, \dots, VM_m\}$ be the set of m virtual machines. The scheduler must ensure a mapping $f: F \rightarrow$ VM such that each task is assigned to one VM for optimization. Table 1 represents an example of 15 tasks and 3 VMs allocations [14].

Table 1: Task-VMs Allocation Representation

Task-VM Mapping Representation						
VM1	T1	T3	T7	T10	T14	-
VM2	T4	T5	T9	T11	-	-
VM3	T2	T6	T8	T12	T13	T15

3.1 Makespan Model

The makespan is the maximum time taken to schedule a group of tasks [13]-[15] which is presented in Eq. (1).

$$\text{Makespan} = \max(F_i) \quad (1)$$

Where, i∈W, W is set or group of all tasks and F_i is finish time of tasks i.

3.2 Cost Model

In real cloud computing, a cloud service provider provides the scheduling and other services on pay-per-usage model. We have designed a cost model [13]-[15] by considering the pay-per-usage concept. It is supposed a data-center has one host holding several VMs utilize the computation cost as given in Eq. (2).

$$cost = \frac{CF+MF}{2} \quad (2)$$

The Migration factor (MF) is computed using Eq. (3). Let's say: 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:

$$MF = \frac{DC}{HM} \times \frac{Migs}{VM_u} \quad (3)$$

Computation cost factor is calculated using Eq. (4). Let's say: 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:

$$CF_{base} = \frac{1}{V}\sum_{j=1}^{V} \frac{PT_j \times Mem_j}{MIPS_j \times DC_{TMips}} \quad (4)$$

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 computed using Eq. (5). Let: w_{sc} is VM scaling weight, then:

$$SF = 1 + w_{SC} \times V^2 \quad (5)$$

Final CF (Cost Factor) is computed using Eq. (6). Let: norm is normalization factor i.e. 1000 to balance the cost for realistic environment.

$$CF = norm \times CF_{base} \times SF \quad (6)$$

3.3 Fitness Function

We have integrated the makespan and cost metrics in a single objective fitness method (F_x) which is calculated using Eq. (7). The number of datacenters and VMs are used to normalize the fitness values. In this research, we are minimizing the values of makespan and cost i.e. minimum fitness indicates better results.

$$F_x = \frac{w_1 \times \text{makespan} + w_2 \times \text{cost}}{DC \times VMs} \quad (7)$$

Where, DC is datacenter, VMs are virtual machines, w_1 and w_2 are the weights used to give the priorities to makespan and cost. Slightly more priority is given to makespan in order to ensure stable performance of the system as end users priority is system's performance. The values of w_1 and w_2 are 0.6 and 0.4 respectively given that $w_1 + w_2 = 1$.

4 Data and Methods

We have used CyberShake, Montage, Inspiral and Sipht workflows data for experiments purposes. Each dataset contains 1000 tasks. The structures of these scientific workflows are available online

(https://pegasus.isi.edu). We have considered these scientific as these are well tested and globally recognized for research purposes.

4.1 HEFT Algorithm

HEFT algorithm as told [13]-[15], is a heterogeneous earliest finish time algorithm with is used to initialize the population initially for all algorithms used in this research. The ranking or priority is assigned to the parent and child tasks available in workflows used in this research. The steps are discussed 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)
```

4.2 Inertia Weight Strategies

In scheduling algorithms, inertia weight strategies are used to improve the balance between exploration and exploitation. Figure 1 is showing three existing inertia weight strategy LDIW (Linearly Decreasing Inertia Weight), OIW (Oscillating Inertia Weight), CIW (Chaotic Inertia Weight), and one proposed HIW (Hybrid Inertia Weight) strategy.

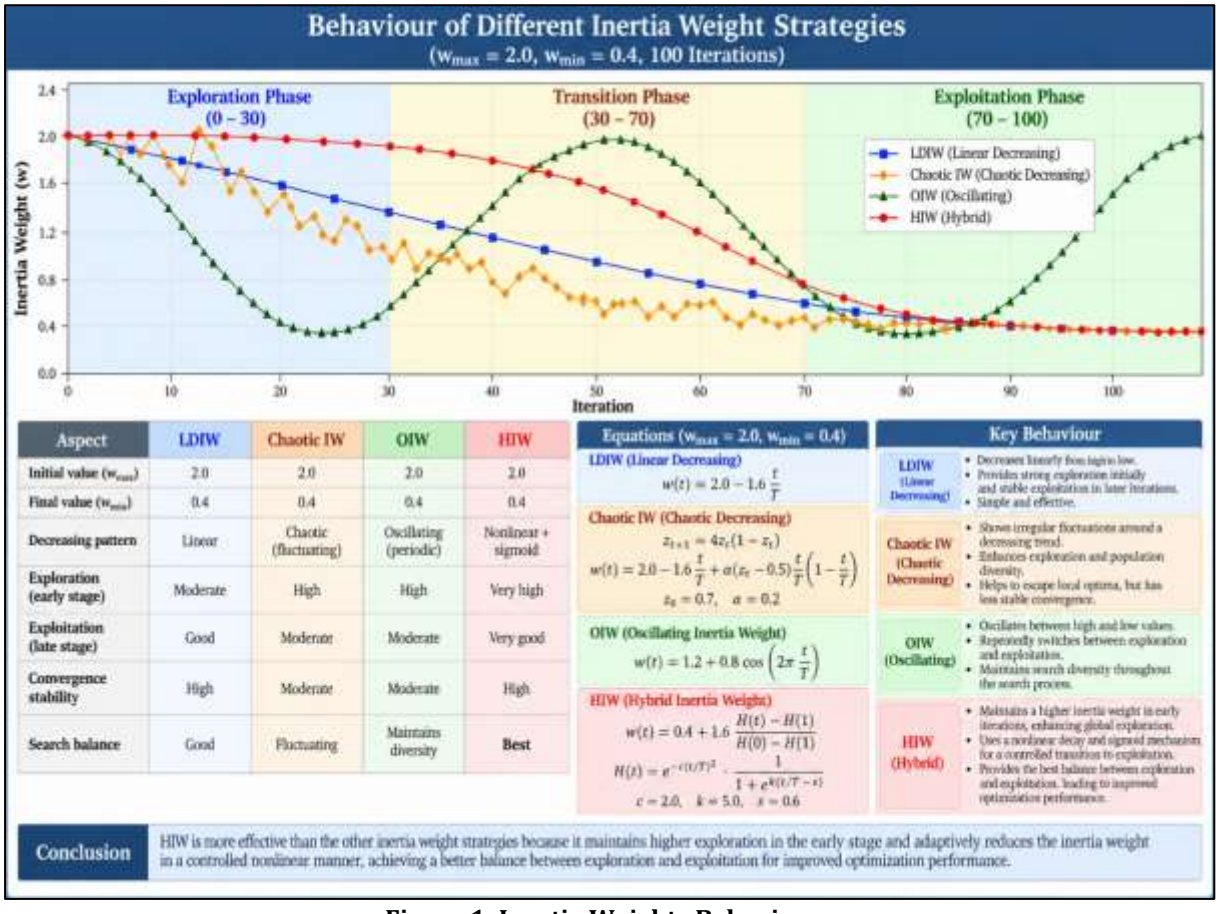

Figure 1: Inertia Weights Behaviour

The behaviour of these LDIW, CIW, OIW and HIW is clearly indicating that HIW maintains a high inertia weight during early iterations and decreases more sharply later through its nonlinear-sigmoid mechanism. The values of different parameters can be seen in Figure 1. Proposed HIW method provides the best exploration and exploitation balance for proposed HICSO algorithm which is guided by HEFT algorithm. The rest story is covered in results and discussion section.

4.3 Proposed Algorithm

In this research Cat Swarm Optimization based algorithm namely HICSO has been proposed. The basic theory of Cat Swarm Optimization is discussed here. Cat Swarm Optimization is a swarm intelligence algorithm was developed by Chu et al. in 2006 [7]. For searching and chasing the food, it has two modes seeking and tracing. These modes are explained below (see Eqs. 8, 9 and 10).

1) Seeking Mode – This mode uses four parameters namely SMP (seeking memory pool), SRD (seeking range dimension), CDC (count dimension to change) and SPC (self-position considerations). A MR (mixing ratio) value is used to distribute the cats in seeking and tracing modes. The SM (seeking mode) can be understood by following steps:
   - Make SMP copies of the Q^{th} cat.
   - Update the position using SRD value given in Eq. (8).
     $$X_{Q,D} = (1 + rand \times SRD) \times X_{Q,D} \quad (8)$$
     Where, $X_{Q,D}$ is cat's position and rand is random variable between (0, 1).
   - Evaluate fitness and find best cat.
   - Swap the original cat with best copy.
2) Tracing Mode – In this mode, the cat travels speedily towards its food by updating velocity and position in order to exploit the target food. The TM (tracing mode) can be understood by following steps:
   - Calculate velocity and position of Q^{th} cat using Eq. (9).
     $$V_{Q,D} = \omega \times V_{Q,D} + \left(c1 \times r1 \times \left(X_{BEST,\ D} - X_{Q,D}\right)\right) \quad (9)$$
     Where, $V_{Q,D}$ is velocity of the cat, c1 is learning coefficient, r1 is a random number between (0, 1), and ω is inertia weight which may be set 0.5 i.e. static in basic algorithm.
   - The position of the cat Q^{th} is updated using Eq. (10).
     $$X_{Q,D} + X_{Q,D} + V_{Q,D} \quad (10)$$
     Where, $X_{Q,D}$ is position of the Cat_Q in dimension D.

The steps involved in this algorithm are shown in Algorithm 2.

Algorithm 2: HICSO Algorithm

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

```
For Gen = 0 to max_Generations Do
    Update Seeking and Tracing Modes using MR value
    For each cat in Cats Do
        If cat is in Seeking Mode Then
            run Seeking Mode
        Else
            // Use Hybrid Inertia Weight in TM
            run Tracing Mode
        End If
        If cat.fitness < gBest.fitness Then
            gBest ← copy (cat)
        End If
    End For
End For
return Best Schedule (gBest)
```

5 Simulation Settings and Result

In this research, CloudSim 3.0 simulator written in Java programming language has been used to conduct the experiments. We have implemented all algorithms 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. We have created 5 types of 70 heterogeneous VMs having computing power between 500-1000 MIPS, 512-1024 MB RAM and 1000 Mbps bandwidth. All algorithms are executed for 15 times to give equal treatment and observe the stability. The rest of the simulation parameters details are defined in Table 2. The performance parameters are makespan and cost described earlier.

Table 2: Simulation Parameters

Proposed HICSO, LDCSO, CICSO and Algorithms Properties	
Parameter	**Value**
Max Generations	200
No. of Cats (Population)	50
SMP, CDC and SRD, MR, c1, r1, ω_{max}, ω_{min}	5, 0.3, 0.3, 0.2 1.5, (0, 1), 2.0, 0.4
CSO Algorithm Properties	
Parameter	**Value**
Generations	200
No. of Cat	50
SMP, CDC and SRD, MR, c1, r1, ω (static inertia weight)	5, 0.3, 0.3, 0.2 1.5, (0, 1), 0.7

5.1 Results Discussion

Table 3 is showing the summary of results obtained after experiments on all four workflows using all algorithms including propose one.

Figure 2 compares HICSO, LDCSO, CICSO, OICSO and CSO algorithms for makespan results. The proposed HICSO algorithm indicates the best scheduling performance. OICSO and LDCSO show moderately higher makespan values. Lower makespan means faster workflows execution and the graph demonstrates the superior performance of HICSO due to a better balance between exploration and exploitation (due to proposed HIW inertia weight used in tracing mode).

Table 3: Results Summary of Makespan and Cost

Workload	**Metric**	**Result Type**	**HICSO**	**LDCSO**	**CICSO**	**OICSO**	**CSO**
CyberShake – 1000	Makespan	Mean	983.368	1232.122	1214.983	1095.451	1817.786
		Best	904.233	1073.027	1019.430	952.986	1090.044
		Worst	1114.921	1394.498	1402.092	1174.205	3077.362
	Cost	Mean	498.554	557.112	533.317	505.746	1112.714
		Best	444.022	515.561	475.919	464.935	532.460
		Worst	539.275	604.967	580.223	556.979	2328.687
Montage – 1000	Makespan	Mean	644.903	735.086	766.953	712.313	1070.831
		Best	622.430	630.474	662.811	626.354	622.147
		Worst	715.011	840.719	872.189	836.084	1578.826
	Cost	Mean	252.188	270.839	272.087	259.802	594.408
		Best	231.206	244.539	239.892	230.722	278.400
		Worst	277.731	309.401	307.954	303.130	1145.735
Inspiral – 1000	Makespan	Mean	16878.782	19724.745	19814.390	18307.582	25386.646
		Best	15976.048	18244.633	18831.982	16184.177	16753.848
		Worst	18109.446	20822.624	21767.819	20058.422	34342.730
	Cost	Mean	4977.179	5610.157	5555.838	5173.878	11251.680
		Best	4635.464	4971.633	4989.909	4662.948	5397.309
		Worst	5556.216	6416.144	6454.632	5549.746	18184.815
Sipht – 1000	Makespan	Mean	20860.403	22749.349	22885.886	21945.302	34247.105
		Best	18733.296	19821.738	20949.939	19311.174	19030.710
		Worst	23185.625	26254.676	25060.554	24810.620	46781.464
	Cost	Mean	3978.420	4448.734	4224.785	4099.493	11958.644
		Best	3571.461	3394.074	3700.404	3492.378	3878.293
		Worst	4611.379	5665.312	5550.042	4973.656	20484.208

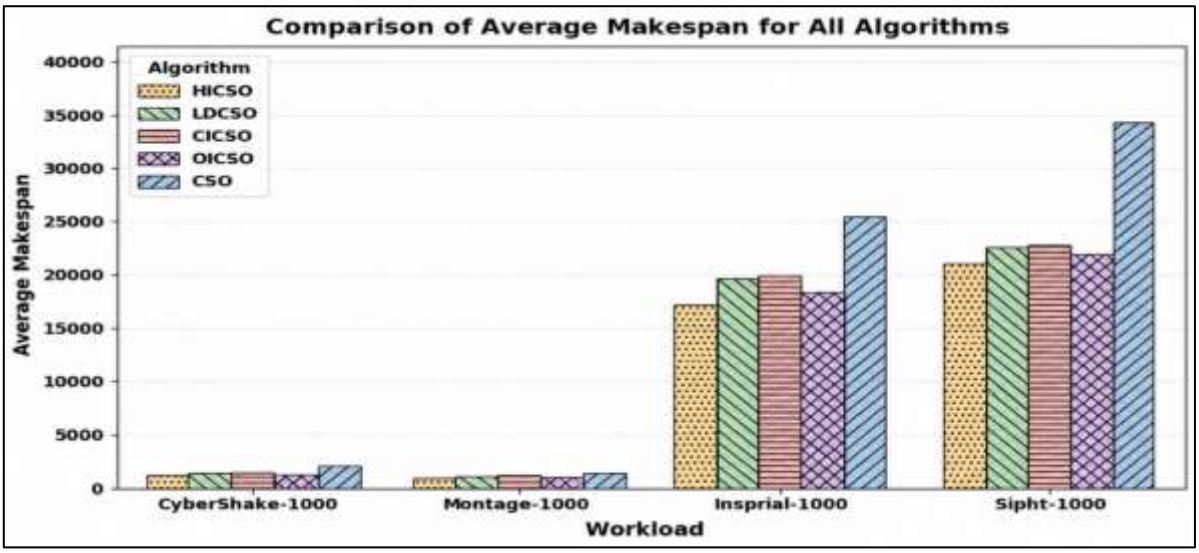

Figure 2: Makespan Comparison

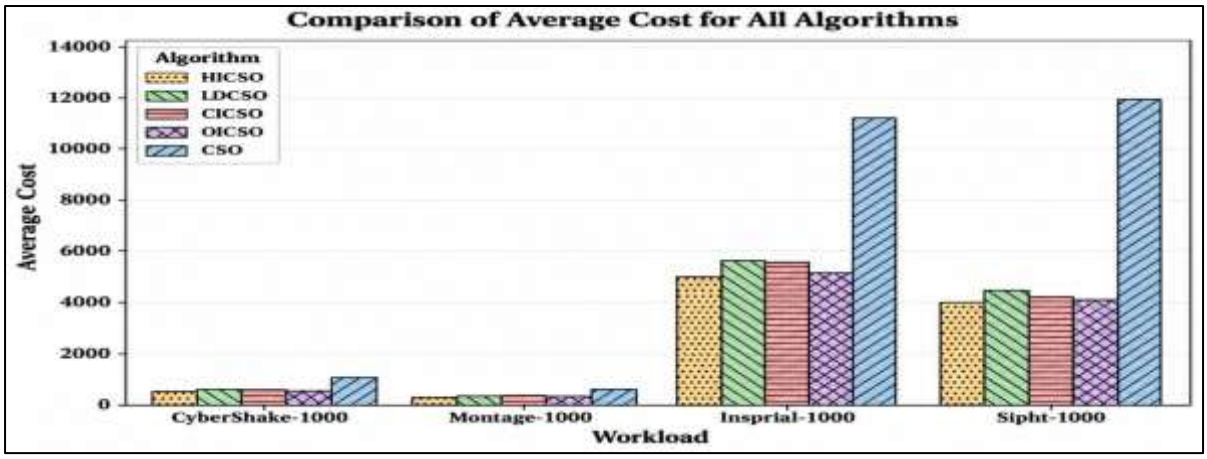

Figure 3: Cost Comparison

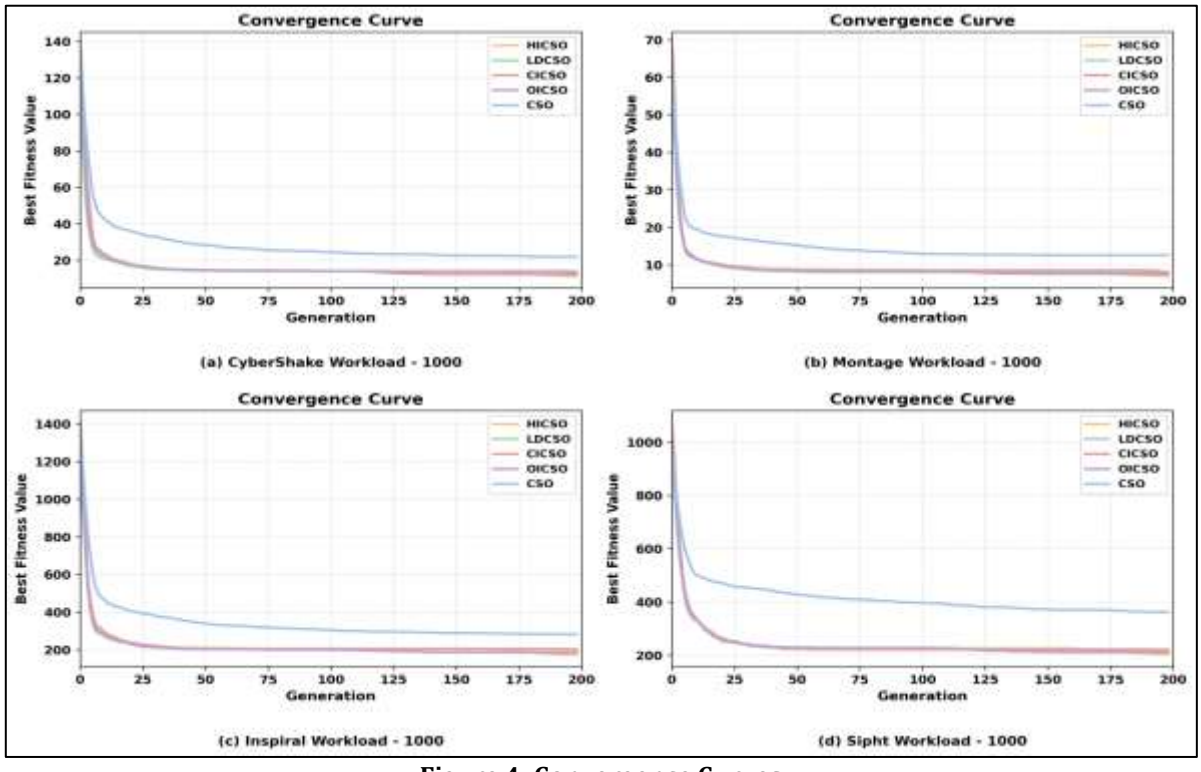

Figure 4: Convergence Curves

Figure 3 compared the cost consumed by the system using all algorithms for all four workflows. The proposed HICSO achieves the lower cost among all algorithms, indicating better cost efficiency. OICSO is also competitive to HICSO in case of makepan and cost. The proposed HICSO algorithm is achieving better results due to good convergence (see Figure 4) and balance between exploration and exploitation. The HICSO achieves 4.94% to 10.23% in makespan and 1.42% to 3.80% in cost over its competitive OICSO algorithm.

6 Conclusions and Future Scope

This study presented HICSO, a hybrid CSO-based workflow scheduling algorithm incorporating the proposed HIW strategy. HIW maintains stronger exploration in the early search stages and performs a sharper nonlinear reduction in later iterations, providing an improved exploration–exploitation balance. Experimental results on CyberShake, Montage, Inspiral, and Sipht workflows demonstrate that HICSO consistently provides the lowest average makespan and cost among the evaluated algorithms. The convergence results further support its effective search and convergence behavior. Overall, HICSO demonstrates strong potential for efficient and cost-effective workflow scheduling in heterogeneous cloud environments.

Further, the algorithms can be tested on other datasets like Google Cluster Traces, Alibaba Cluster Traces and others. A hybrid or extended version of the HICSO algorithm can also be developed.

References

1. J. Kennedy and R. C. Eberhart, “Particle swarm optimization,” *Proceedings of the IEEE International Conference on Neural Networks*, vol. 4, pp. 1942–1948, 1995, doi: 10.1109/ICNN.1995.488968.
2. Y. Shi and R. C. Eberhart, “A modified particle swarm optimizer,” *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, pp. 69–73, 1998, doi: 10.1109/ICEC.1998.699146.
3. S. Zhao, X. Fu, H. Li, G. Dong, and J. Li, “Research on cloud computing task scheduling based on improved particle swarm optimization,” *International Journal of Performability Engineering*, vol. 13, no. 7, pp. 1063–1069, 2017, doi: 10.23940/ijpe.17.07.p8.10631069.
4. Q. Wang, X.-L. Fu, G.-F. Dong, and T. Li, “Research on cloud computing task scheduling algorithm based on particle swarm optimization,” *Journal of Computational Methods in Sciences and Engineering*, vol. 19, no. 2, pp. 327–335, 2019, doi: 10.3233/JCM-180874.
5. Z. Zhou, J. Chang, Z. Hu, J. Yu, and F. Li, “A modified PSO algorithm for task scheduling optimization in cloud computing,” *Concurrency and Computation: Practice and Experience*, vol. 30, article e4970, 2018, doi: 10.1002/cpe.4970.
6. X. Huang, C. Li, H. Chen, and D. An, “Task scheduling in cloud computing using particle swarm optimization with time varying inertia weight strategies,” *Cluster Computing*, vol. 23, no. 2, pp. 1137–1147, 2020, doi: 10.1007/s10586-019-02983-5.
7. S. C. Chu, P.-W. Tsai, and J.-S. Pan, “Cat swarm optimization,” in *PRICAI 2006: Trends in Artificial Intelligence*, LNAI 4099, pp. 854–858, Springer, 2006, doi: 10.1007/11801603_94.
8. S. Bilgaiyan, S. Sagnika, and M. Das, “Workflow scheduling in cloud computing environment using cat swarm optimization,” in *Proceedings of the 2014 IEEE International Advance Computing Conference (IACC)*, pp. 680–685, 2014, doi: 10.1109/IAdCC.2014.6779406.
9. S. Bilgaiyan, S. Sagnika, and M. Das, “A multi-objective cat swarm optimization algorithm for workflow scheduling in cloud computing environment,” in *Intelligent Computing, Communication and Devices*, pp. 73–84, Springer, 2014, doi: 10.1007/978-81-322-2012-1_9.
10. D. Gabi, A. S. Ismail, and N. M. Dankolo, “Minimized makespan based improved cat swarm optimization for efficient task scheduling in cloud datacenter,” in *Proceedings of the 2019 3rd High Performance Computing and Cluster Technologies Conference (HPCCT 2019)*, pp. 16–20, ACM, 2019, doi: 10.1145/3341069.3341074.
11. D. Gabi, N. M. Dankolo, A. S. Ismail, A. Zainal, and Z. Zakaria, “Non-preemptive chaotic cat swarm optimization scheme for task scheduling on cloud computing environment,” *International Journal of Advanced Computer Research*, vol. 9, no. 43, pp. 186–196, 2019, doi: 10.19101/IJACR.PID29.
12. S. Mangalampalli, S. K. Swain, T. Chakbrabarti et al., “Prioritized task-scheduling algorithm in cloud computing using cat swarm optimization,” *Sensors*, vol. 23, no. 13, article 6155, 2023, doi: 10.3390/s23136155.
13. J. Bhagwan, and S. Kumar, “An HC-CSO algorithm for workflow scheduling in heterogeneous cloud computing system,” *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 6, pp. 484-492, 2021, doi:10.14569/IJACSA.2021.0120655.

14. J. Bhagwan, and S. Kumar, "An H-CSO algorithm for workflow scheduling in heterogeneous cloud environment," *International Journal of Intelligent Engineering & Systems*, vol. 14, no. 5, pp. 422-434, 2021, doi: 10.22266/ijies2021.1031.37.
15. Jai Bhagwan, S. Kumar et al. "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. 3s, pp. 399-412, 2026, doi: 10.51483/IJAIML.6.2s.2026.399-412.
16. J. Bhagwan, Majon, S. Kumar, S. Godara and S. Seema, "SACSO: an intelligent algorithm for energy efficient workload scheduling in cloud computing, *International Journal of Artificial Intelligence and Machine Learning*, vol. 6, no. 7s, pp. 262-272, 2026, doi: 10.51483/IJAIML.6.7s.2026.262-272.