SvedbergOpen
DISSEMINATION OF KNOWLEDGE
International Journal of Artificial Intelligence and Machine Learning
Publisher's Home Page: https://www.svedbergopen.com/
Research Paper Open Access

IJPSO: An Improved Hybrid PSO Algorithm For Multi-Type And Large Scale Data Scheduling In Cloud Computing

Jai Bhagwan[1]*, Seema Rani[2], Manoj[1], Sunila Godara[1], Yashasvi[1], Sanjeev Kumar[1]

[1]Department of Computer Science & Engineering, Guru Jambheshwar University of Science & Technology, Hisar – 125001, India
[2]Departement of Computer Science & Engineering, Ch. Devi Lal State Institute of Engineering & Technology, Sirsa, India
*Corresponding Author E-mail Address: drjaicse@gmail.com

Abstract - The services of cloud computing are expanding quickly. The objective of cloud computing is to offer online services as it is a pay-per-use model through which users can manage data as per their needs. The number of cloud users is growing, so task scheduling is considered an important issue for allocating tasks to resources. In recent years, numerous nature-inspired meta-heuristic algorithms have been introduced and can be applied to solve task scheduling issues. However, these techniques do not consider simultaneously multi-type workload scheduling such as workflows and independent tasks. In this research, various task scheduling algorithms have been examined and a new algorithm has been designed named IJPSO using JAYA and particle swarm algorithms to enhance the cloud system performance. PSO suffers from local stagnation issue; therefore, JAYA acts as a local refinement operator. Adaptive inertia weight is adopted to balance the local and global searching capabilities. Simulations were performed using scientific workflows and independent Google traces 2019 dataset. The newly designed algorithm has performed better than Particle Swarm Optimization (PSO), Grey Wolf Optimization (GWO) and IPSO in terms of makespan and monetary cost.

Keywords – Cloud Computing, Cost, JAYA Algorithm, Makespan, Particle Swarm Optimization, Virtual Machines (VMs).

1. Introduction

Cloud computing is evolving technology in the era of information technology which is most aggressively used in various fields. It enables the consumer to maximize their productivity potential by providing unified and scalable services. Cloud system is used by billions of users worldwide. In response to customer demands, the cloud offers immediate and universal access to shared resources which are quickly efficient and affordable (S. T. Milan et al., 2019). One of the core reasons behind the growth of cloud is its reliability, scalability and efficiency without worrying about hardware requirements. Virtualized services such as IaaS (Infrastructure as a Service), PaaS (Platform as a Service), and SaaS (Software as a Service) are available in cloud technology in terms of pay-per-use model (Hussain et al., 2017). 'Infrastructure as a service' provides on-demand access to virtualized resources like storage, infrastructure, machines, computing, and communication. These resources are easily virtually scalable according to consumer demands. The IaaS model provides the physical servers to customers in virtual form instead of purchasing of costly hardware. The PaaS provider offers tools and development services, typically the tools required by the developers to build, publish, and execute their own applications. The cloud providers should establish the essential cloud infrastructure, including network, servers, operating system, and storage. It enables connections between databases and web services. Using virtualization technology, resources can be easily scaled up or down to meet business needs. Software as a Service (SaaS) is a service model that provides software to the users which is easily accessible to them via internet. In order to create and deploy contemporary applications, SaaS services are used. As the the number of cloud users increasing rapidly, traffic on cloud servers is increasing continuously (Alouffi et al., 2021).

Task scheduling is crucial in managing the execution of tasks among the resource consumption which enhances performance and lowers costs while managing the large scale load on cloud resources (Ebadifard and Babamir, 2018).
The term "task scheduling" refers to the distribution of tasks among virtual machies (Mapetu et al., 2019) in an efficient way. The scheduling problem is considered as NP-hard (Mansouri et al., 2019) problem. A crucial component of task scheduling is load balancing which enables the assignment of a high volume of heterogeneous incoming requests to heterogeneous resources in accordance with requirements (Hemasian-Etefagh and Safi-Esfahani, 2019). Load balancing is a technique to identify under-loaded and overloaded VMs and distribute the load among the virtual machines to balance the workload (Ebadifard and Babamir, 2018; Pradhan et al. 2022). It should be ensured that no virtual machine be idle, under-loaded or over-loaded (Ngatman et al., 2017). The prime goal of load balancing is to distribute the workload which can minimize one or more parameters considered as completion time, resource utilization rate, degree of imbalance, makespan etc (Singh et al., 2019). This can be achieved by using task scheduling algorithms which provide near-optimal solution. Many task scheduling algorithms have been designed so far to minimize makespan, cost, energy, degree of imbalance etc. Despite these advancements, gaps remain in conducting big data scheduling and real-time mobility. The main contributions of this research paper are:

- Designed mathematical models for makespan and cost.
- Designed a priority-weight based fitness function considering makespan, cost and degree of imbalance.
- Introduced HEFT to maintain parent-child dependency for scientific workloads and to set priorities for independent tasks.
- Proposed an IJPSO algorithm for makespan and cost aware efficient scheduling while improving the throughput.
- Introduced inertia weight named SMIW in proposed algorithm to balance global and local searching gradually and injected it in velocity update formula of IJPSO.
- Injected JAYA algorithm in IJPSO to enhance the local searching refinement capability in order to find more efficient solutions.
- Performance comparison between the proposed IJPSO, IPSO, PSO, and GWO demonstrate its effectiveness.

Rest of the paper is presented into the following sections. The literature review is described in section 2. In section 3, system model and problem design is discussed. The algorithm design is explored in section 4. The simulation environment and results discussion are presented in section 5. Finally conclusion and future work directions are explained in section 6.

2. Literature Review

Heuristic algorithms can generate feasible solutions under specific constraints. Hence, solution obtained relies heavily on set of rules and also the size of the problem which makes it more expensive and absurd. Therefore, it is better to use the meta-heuristic algorithms for task scheduling in cloud computing. The widely used algorithms to solve scheduling problems are described here.
Saleh et al. (2019) proposed an Improved PSO for large scale task scheduling. First, the tasks were sliced into different groups in a dynamic way for efficiency. Load balancing mechanism along with PSO improved the performance compared to other state of arts algorithms. Mansouri et al. (2019) designed a hybrid task scheduling algorithm namely FMPSO based on modified PSO and fuzzy theory. At first, the global search capacity was improved using modified velocity and roulette wheel method. To avoid local optima issue, crossover and mutation operators were used. The proposed algorithm improved makespan, degree of imbalance, and total execution time compared to other algorithms. Alzaqebah et al. (2019) claimed that Modified-GWO has proved to be a successful modified approach based on GWO algorithm. Primary goals of this approach were cost and makespan reduction. The results obtained using this technique are better than traditional GWO and WOA with regard to makespan, cost and the extent of imbalance. Gohil et al. (2018) proposed IGWO which is inspired by the hunting technique and social hierarchy of the wolves. Several experiments were conducted in terms of an average value of objective function. It was observed to be successful algorithm due to its ability to obtain near-global optimal solutions and minimizing local minima stagnation. It has the ability to solve real-world optimization issues (Laabadi et al., 2020) as well. The algorithm was compared with ABC, PSO, and GWO. An improved grey wolf optimization algorithm was proposed in to optimize cost and time by (Natesan et al., 2020) namely PCGWO. It is found responsible for reducing both processing time as well as cost during task scheduling. Xingjun et al. (2020) designed GWO based approach and demonstrated its effectiveness in addressing the NP-hard task scheduling problem. The proposed method reduced the response time and improves the degree of load balancing compared to TSLBACO, HJAS. Natesan and Chokkalingam (2019) presented a Mean-GWO method based on grey wolf

behavior which is used to resolve the issue of task scheduling by minimizing the makespan, energy consumption. The results demonstrate that the mean-GWO method performs better than PSO and standard GWO. Chen et al. (2019) introduced Improved WOA to increase the effectiveness of task scheduling. The paper discussed the improvement of convergence speed and accuracy of the WOA based approach. After comparison with ACO and PSO the proposed method was found effective.

Luo et al. (2018) introduced an adaptive-weight based algorithm, which helps in improving efficiency of resources and reduce the completion time. It performs better compared to standard PSO. The PSO gets stuck in late stages in local optima. The weights changes occur during iterations for avoiding local optima stagnation. Wu (2018) introduced an iterative selection operator which was used to solve the task scheduling problem. The results show that it improves completion time efficiency. Awad et al. (2015) designed a LBMPSO algorithm to contribute to improving the reliability. It was compared with the standard PSO, a random scheduling algorithm and longest cloudlet to fastest processor. Kumar et al. (2021) designed an Enhanced CSA algorithm for cloud computing to solve NP-hard problems. Its effectiveness was evaluated in comparison to CSA and Max-Min algorithm. The makespan and degree of imbalance were reduced compared to max-min and CSA. Mishra et al. (2022) proposed a binary JAYA method for task mapping onto virtual machines (VMs). It improves resource utilization, performance and reduces the energy consumption. Jena et al. (2021) designed a binary JAYA algorithm to evaluate the performance. The suggested approach was compared against the standard JAYA, PSO, SSA and BSO. The simulation results show significant improvements in response time, reduced degree of imbalance, makespan while improving resource utilization.

From the literature, it is identified that various methods have been developed to solve the issue of task scheduling. Every swarm intelligence algorithm has either an exploration or an exploitation problem. We consider Particle Swarm Optimization algorithm which is easy to implement and can solve various problems including task scheduling effectively. The PSO algorithm suffers from stagnation in local optima. Although, various versions of PSO have been developed so far, they do not address the multi-type workload scheduling problem in the era of IoT. So, in this research we have overcome the weak exploration issue of PSO as well as the balance between local and global searching. Our proposed IJPSO algorithm solves the multi-type workload scheduling efficiently and faster convergence speed compared to other methods.

3. System Model and Problem Design

Problem formulation, makespan and cost models are described in this section. In cloud computing task scheduling, we consider how a scheduler assigns tasks to available virtual machines. We are optimizing makespan and monetary cost in this research.

3.1 Problem Definition

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 determine a mapping $f: F \rightarrow VM$ such that each task is assigned to one VM. The example can be seen with 15 tasks and 3 VMs in Table 1.

Table 1: Task-VM Mapping Representation

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

3.2 Makespan Model

In cloud computing task scheduling, the total completion time taken by all tasks in a workflow or task set is known as makespan. Its mathematical model is given in Eq. (1).

$$Makespan = max(F_i) \quad (1)$$

Where, i∈T, T is set of all tasks and F_i is finish time of tasks i.

3.3 Financial Cost Model

The cloud computing service provider offers the services on pay-per-use model. Keeping in mind this pay-per-use concept, we have designed a cost model (Khurana and Singh, 2019; J. Bhagwan and S. Kumar, 2021). It is supposed a data-center contains one host and several VMs utilize the financial computation cost as given in Eq. (2).

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

The Migration factor (MF) is computed using Eq. (3). Let: HT is number of host machines, DC is number of data-centers, Migs is number of tasks migrations and VM_u is number of VMs used, then:

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

The computation cost factor is computed using Eq. (4). Let: V is a set of VMs, PT_j is processing time on VM_j, $Memory_j$ is total task memory on VM_j, $MIPS_j$ is MIPS of VM_j and $DC_{TotMips}$ is datacenters' MIPS, then:

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

Here, 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_{scale} is VM scaling weight, then:

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

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

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

3.3 Throughput

In this research, we have used the throughput (Zhao, 2024) metric to observe the system's performance; it is computed using Eq. (7). The throughput metric has not been included in the fitness function and it has been calculated separately to avoid complex fitness calculation for a scheduler.

$$throughput = Total\ Tasks/Makespan \quad (7)$$

3.4 Fitness Function

To optimize the makespan and cost we have combined these in a single objective fitness function (F_x) which is discussed in Eq. (8). We have used number of DC and VMs to normalize the fitness values. Also, we are minimizing the values of fitness function during the number of iterations to optimize the makespan and cost.

$$F_x = \frac{w_1 \times makespan + w_2 \times cost}{DC \times VMs} \quad (8)$$

Where w_1 and w_2 are the weights used to give the priorities to makespan and cost. Here, we are giving slightly more priority to makespan to stable the system performance. The values of w_1 and w_2 are 0.6 and 0.4 respectively given that $w_1 + w_2 = 1$.

4. Methods and Workload Data

This section describes the proposed algorithms in detail. Our research is based on Particle Swarm Optimization, SMIW inertia weight and JAYA Algorithm.

4.1. Basic Particle Swarm Optimization

The particle swarm optimization is a swarm intelligence family algorithm which is a part of artificial intelligence. The algorithm was introduced by (J. Kennedy and R. Eberhart, 1995). This algorithm is inspired by the behaviour of birds flocking and fish schooling. The mathematical model of the PSO algorithm is given in Eq. (9) and (10). The candidate solutions called particles move though the search space and each particle remembers its personal best position call pBest. The best solution found by the entire swarm is called as gBest. For moving toward better solutions, the particles continuously update their velocity and position (Saleh et al., 2019).

The mathematical model of velocity update is given in Eq. (9).

$$v_i^{t+1} = wv_i^t + c_1 r_1 (pBest_i - x_i^t) + c_2 r_2 (gBest - x_i^t) \quad (9)$$

The mathematical model for position update is given in Eq. (10).

$$x_i^{t+1} = x_i^t + v_i^{t+1} \quad (10)$$

Where, w is inertia weight, c_1 is cognitive coefficient, c_2 is social coefficient, r_1 and r_2 are random numbers in [0, 1], x_i is current position and v_i is current velocity.

4.2 JAYA Algorithm

JAYA is a population based optimization algorithm introduced by (Rao, 2016). The word JAYA means "victory" in Sanskrit language. The algorithm tries to move solutions towards the best solution and away from the worst solutions during its evolution. The mathematical model (Jena et al., 2021; Mistra et al., 2022) of the JAYA algorithm is given in Eq. (11).

$$x_{i,j}^{new} = x_{i,j} + r_1\left(Best_j - \left|x_{i,j}\right|\right) - r_2(Worst_j - \left|x_{i,j}\right|) \quad (11)$$

Where, $x_{i,j}$ is current value of solution i in dimension j, $Best_j$ is best solution value, $Worst_j$ is worst solution value, r_1 and r_2 are random values in [0, 1].

4.3 Self-Motivated Inertia Weight

The Self-Motivated Inertia Weight (ω) (Ting et al, 2012; Chen et al., 2014) has been used for newly proposed IJPSO. It is given in the Eq. (12) for velocity update so that exploration and exploitation can be balanced.

$$\omega = \omega_{max} \times exp\left(-a \times \left(\frac{itr}{itr_{max}}\right)^{b}\right) \quad (12)$$

Where, ω_{max} is the initial inertia weight set to 2.0. a is the local searching attractor and b is the global searching attractor.

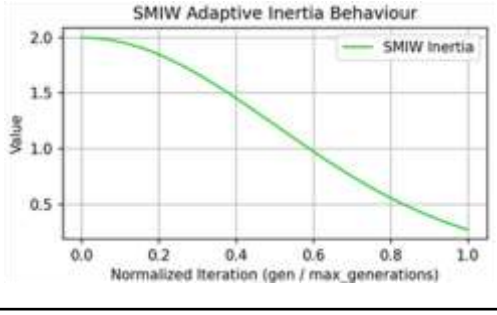

Figure 1: SMIW Decay Behavior

The local and global attracters are used to balance the local and global searching respectively. Basically b controls the decay rate with the iterations gradually. Larger value of b than a leads to exploitation and smaller value leads to exploration. So, changing of ω over iterations is decisive. Here, ω decreases with iterations' incremental process as ω formula defines the exponential decay function. a and b are set to 2.0 to focus on balancing the exploration and exploitation. This inertia weight overcomes the problem of the balance between local and global searching. In early stages, it allows searching optimal solutions locally and then moves towards exploitation or global best. The example of gradually decreasing behaviour is shown in Fig. 1.

4.4 HEFT algorithm

The Heterogeneous Earliest Finished Time algorithm is very famous to calculate the earliest finished time and maintain the DAG dependency as well. This HEFT algorithm can work for both dependent and independent tasks.

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 = 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)
```

In case of independent tasks, the dependencies are considered nil and the average earliest finish time is calculated for the ranking i.e. priorities purposes whereas, in case of workflows the dependencies are included, the process is shown in Algorithm 1.

4.5 Proposed IJPSO Algorithm

The proposed algorithm combines the strength of JAYA and PSO algorithm to obtain the efficient workflows and independent tasks scheduling. The initial population is generated by HEFT policy, which provides a good starting point for optimization. The step by step explanation is given as:

1. After submitting the initial refined tasks by HEFT policy, calculate the SMIW (Self-Motivated Inertia Weight) ω using Eq. (12). The calculated inertia weight is used in velocity update formula given in Eq. (9) for smooth searching decay i.e. moving towards local to global searching gradually. Higher value of inertia weight encourages the global search and lower value promotes local refinement.
2. Identify the Best Particle and Worst Particle from the current population according to their fitness values.
3. Update PSO velocity and position and apply the JAYA update. The JAYA component moves the solution towards the best particle and away from the worst particle (see line 8).
4. Generate the hybrid position using (see line 9). The weighted combination preserves the PSO search capability while incorporating JAYA's guiding mechanism.
5. Evaluate the fitness of updated particle and update personal and global best.
6. Repeat the process until the maximum number of generations is reached.
7. Finally, return best solution.

The detailed steps of the proposed IJPSO algorithm are also given in Algorithm 2.

Algorithm 2: Proposed IJPSO Algorithm

Input: Generate Populations pop by HEFT policy, c_1, c_2, r_1, r_2, r_3, r_4, max_Generations etc.
Output: Best Schedule gBest

```
1.  For Gen = 0 to max_Generations Do
2.      Calculate SMIW inertia weight ω using Eq. (12)
3.      Find BestParticle and WorstParticle in pop
4.      For each particle Pi in pop Do
5.          For each dimension d Do
6.              Update velocity using Eq. (9) // (use SMIW weight)
                // Update positions
7.              psoPos ← Pi.position[d] + Pi.velocity[d]
8.              jayaPos ← psoPos + r3×(BestParticle[d] - |psoPos|) – r4× (WorstParticle[d] - |psoPos|)
                // Hybrid JAYA-PSO Position
9.              Pi.position[d] ← 0.3 × jayaPos + 0.7 × psoPos
10.         End For
11.     Evaluate fitness(Pi)
12.     update pBest and gBest values
13.     End For
14. End For
15. return Best Schedule (gBest)
```

The impact of JAYA component significantly enhances the conventional PSO. The ordinary PSO may get stagnated in local optima due to which premature convergence occurs. JAYA algorithm introduces an additional learning mechanism. It allows moving towards best solution which accelerates convergence near promising regions. The JAYA component also allows moving away from worst solution, this repulsion method prevents particles from remaining low-quality search areas. The JAYA algorithm continuously pushes particles away from poor solutions and reducing the risk of local optima. PSO explores different VM assignments whereas JAYA refines assignments towards high quality schedules. Thus, JAYA acts as a local refinement operator in PSO algorithm.

4.6 Workload Data

The workloads used for scheduling in this research are CyberShake and Inspiral workflows, each containing 100 and 1000 tasks; Google Cluster Traces 2019 based independent tasks ranging from 100-1400 tasks. The workflows are a combination of parent and child dependent tasks, the structure of these are available on https://pegasus.isi.edu and the Google Cluster 2019 Traces are available on Kaggle repository in the form of CSV.

5. Simulation Setup and Results Analysis

The simulation is performed in CloudSim 3.0 tool which is written in Java programming language. The number of Datacenter used in this research is one containing one host machine. Rest, table 2 is describing the simulation environment.

Table 2: Simulation Environment

Parameter	Value
No. of VMs	50
VMs RAM	512-1024
Computing Power	500-1024 MIPS
RAM	512-1024 MB
Bandwidth	1000 Mbps
PEs (CPUs)	1 For each VM
PSO, IPSO and Proposed IJPSO Algorithms Properties	
No. of Population	100
No. of Generations	250
C_1, C_2, r_1 and r_2	1.5, 1.5, r_1 and r_2 random in [0, 1]
GWO	
No. of Population	100
No. of Generations	250

5.1 Performance Metrics

The performance evaluation metrics are makespan, financial cost and throughput which are already discussed in section 3.

5.2 Results and Discussion

In this research, we have worked with two scenarios.

1) With scientific workflows workloads
2) Google Cluster 2019 traces independent workloads

In first scenario two major scientific workflows containing 100 and 1000 tasks are tested namely CyberShake-100, CyberShake-1000, Inspiral-100 and Inspiral-1000.

Figure 2 presents the performance comparison of proposed IJPSO, IPSO and GWO on CyberShake workloads. Figure 2 (a) is indicating that the makespan of all algorithms increases with the number of tasks increments due to higher computational demand. The proposed IJPSO algorithm consistently achieves lower makespan especially for 1000 tasks. It is showing its ability to generate more efficient schedules for a large scale problem. The ability of achieving good performance for large scale problem is due to the integration of JAYA search mechanism which guides particles towards better solutions while avoiding worst ones.

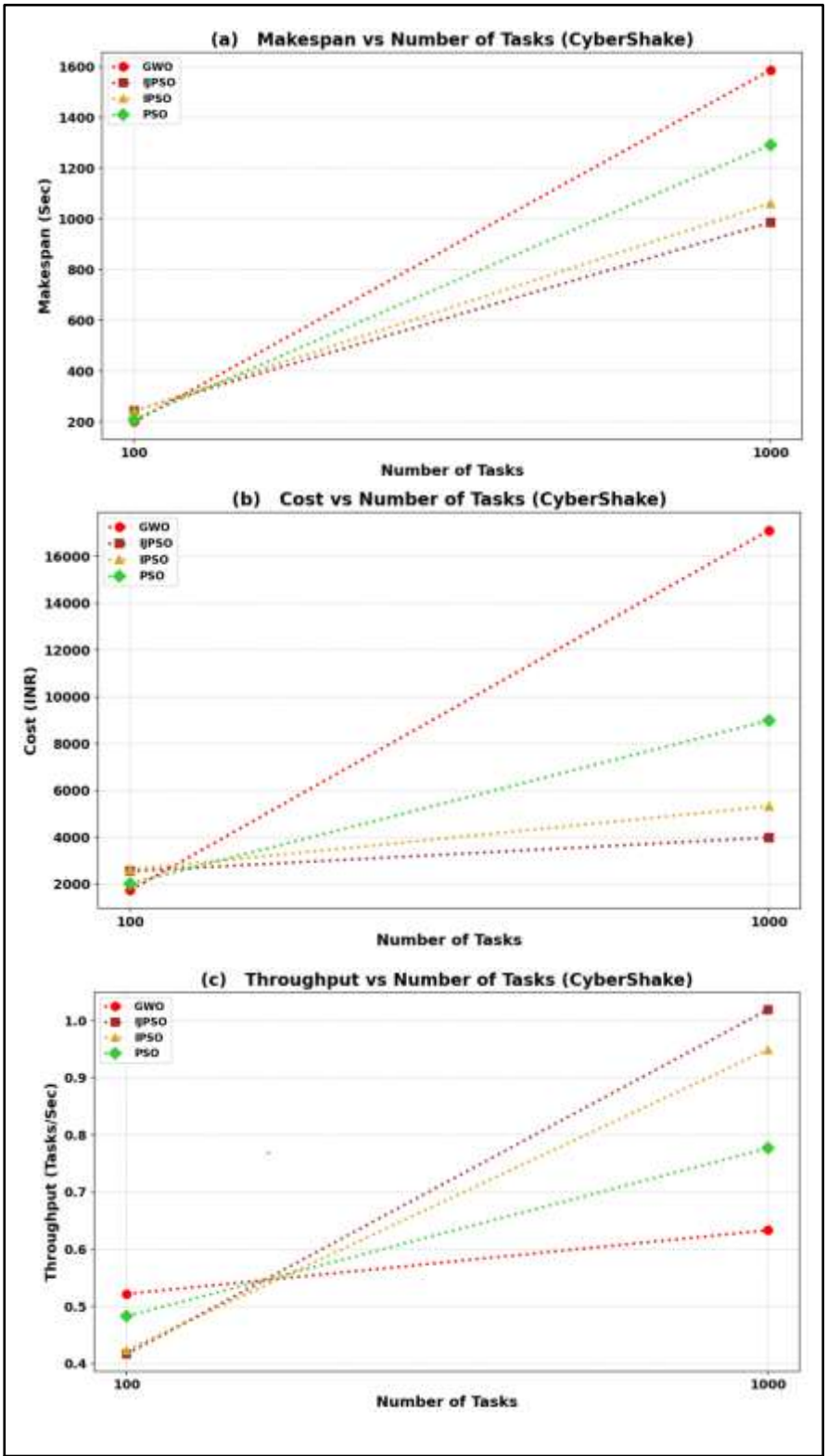

Figure 2: CyberShake Workload Results

Figure 2 (b) illustrates the execution cost consumed by each algorithm during scheduling. Although, the cost is increased with the workload size for all methods due to high computing demand but the proposed IJPSO algorithm took lower cost compared to other algorithms. The PSO and IPSO provide moderate improvements and the GWO performs worst compared to others. The efficiency of IJPSO demonstrates that the algorithm is utilizing cloud resources efficiently due to which it consumes lowest cost compared to others for large scale workload.

Fig. 2 (c) compares the throughput achieved by all algorithms including proposed IJPSO. For large scale workload, the best throughput is generated by our proposed IJPSO algorithm. The second best is IPSO and the worst one is GWO in this large scale research. The higher throughput of the IJPSO algorithm is the consequence of its reduction in makespan and better resource utilization strategy.

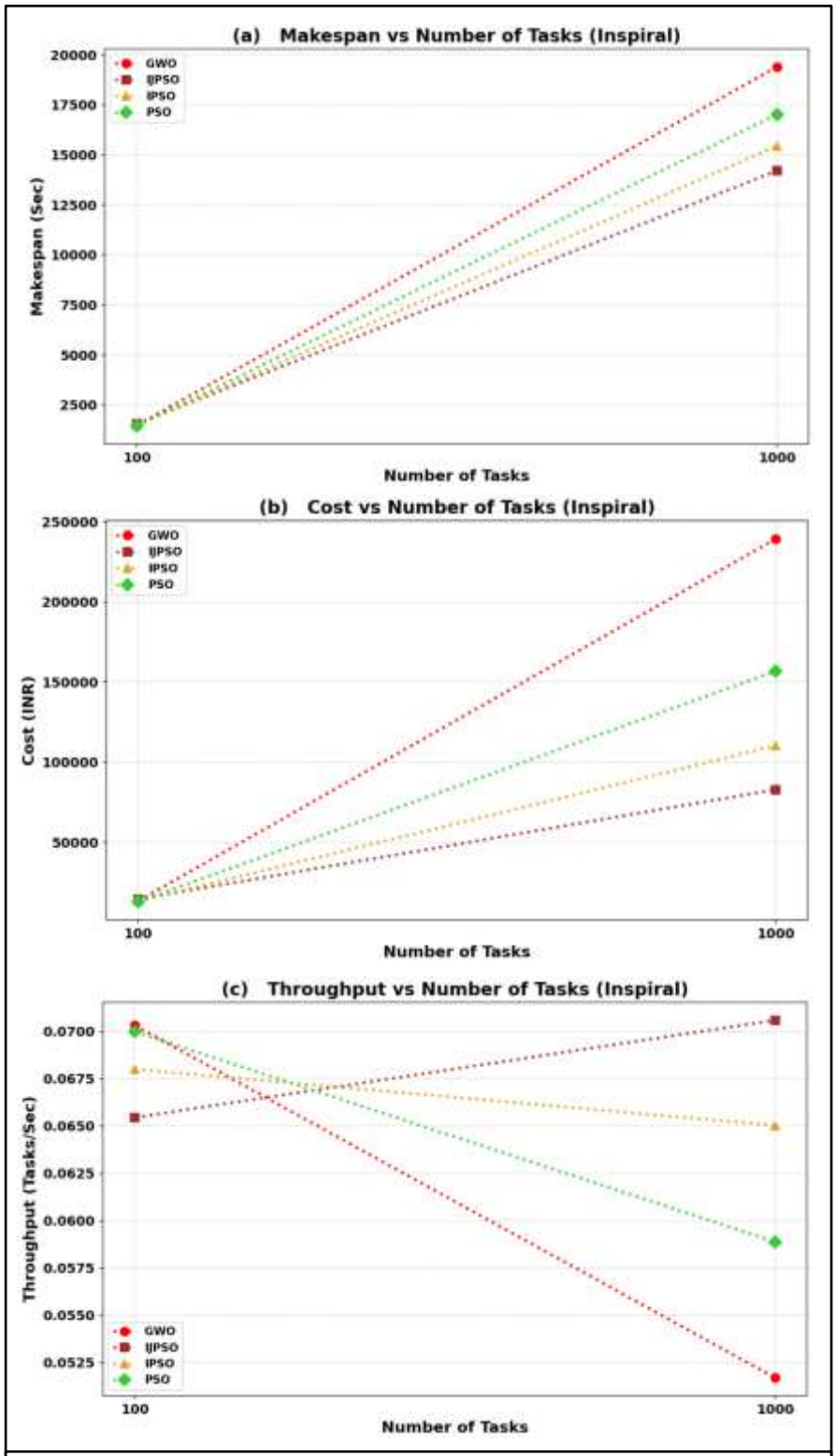

Figure 3: Inspiral Workload Results

Figure 3 presents the performance of all algorithms on Inspiral workflows workloads. The comparison is based on three important scheduling metrics makespan, cost and throughput. The proposed IJPSO algorithm also achieves better makespan, cost and throughput compared to IPSO, PSO and GWO algorithms. As illustrated in Figure 3 (a), the makespan for all algorithm increases for large scale workload, but the proposed IJPSO algorithm gives superior performance by achieving lower makespan. At 1000 tasks, GWO records highest makespan followed by PSO and IPSO compared to our proposed IJPSO multi-type workload scheduling algorithm. This is because of JAYA fusion, the JAYA operator provide efficient resource utilizations and enhance convergence speed for good solutions.

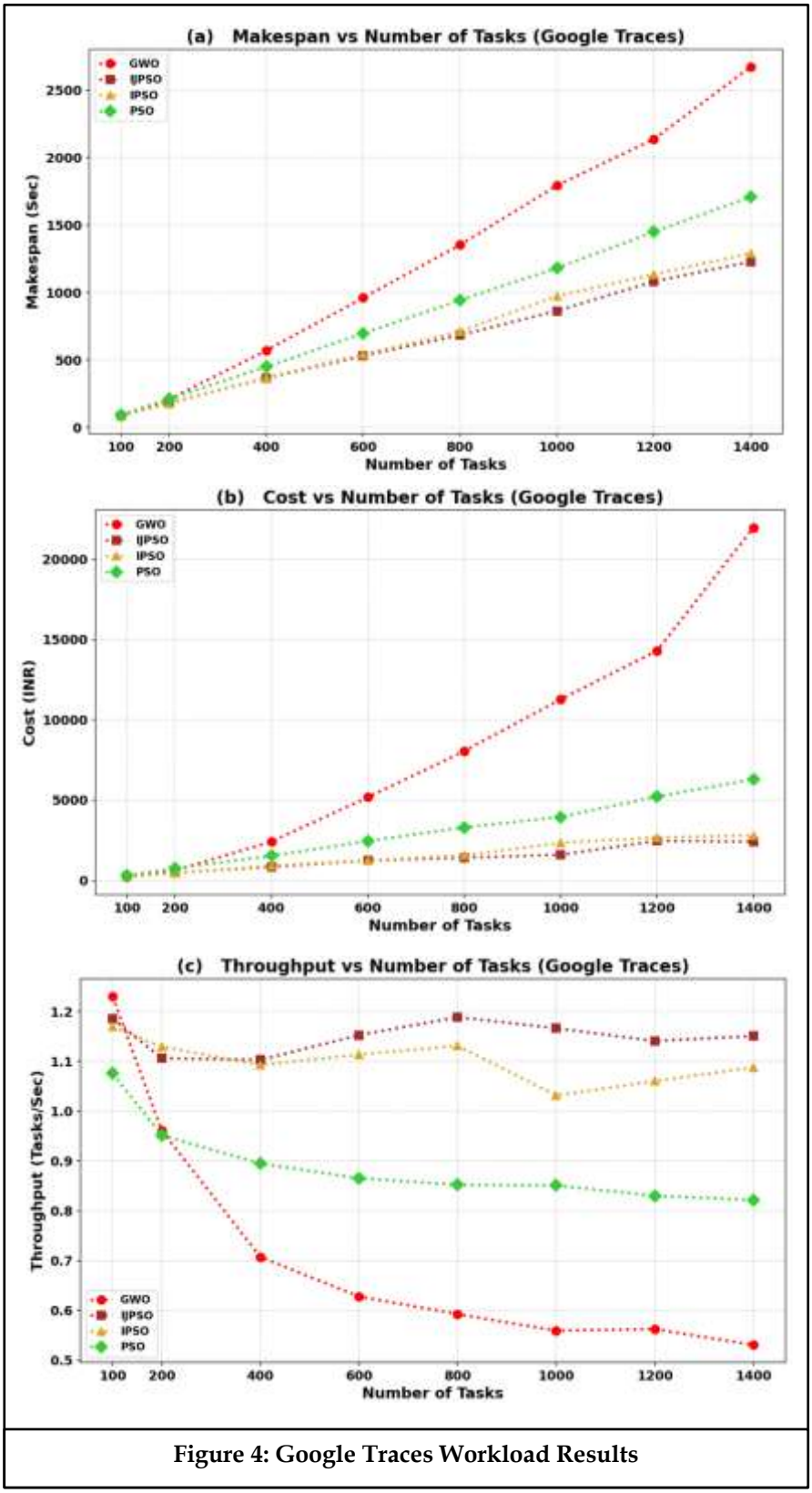

Figure 4: Google Traces Workload Results

Figure 3 (b) demonstrate the incurred financial cost by all algorithms. The cost rises for large scale workload particularly for GWO and then PSO. The IJPSO reduced the overall expenditure among all competing algorithms. This indicates that the IJPSO is capable to select the resources efficiently due to JAYA local refinement and good convergence.
The throughput comparison is presented in Figure 3 (c). For small tasks groups, all algorithms produce almost same throughputs. But in case of larger scale workload, the proposed IJPSO algorithm outperforms IPSO, GWO and PSO. The increase in throughput confirms that the IJPSO algorithm can process a greater number of tasks within almost same execution period due to its improved scheduling decisions.

Table 3: Results Summary Workflows Workload

Workload	Metric	Result Type	GWO	IJPSO	IPSO	PSO
CyberShake 100	Makespan	Mean	199.92	244.34	243.85	209.8
		Best	147.55	181.44	181.74	176.68
		Worst	328.1	334.12	377.95	273.22
	Cost	Mean	1728.35	2550.35	2617.12	2023.7
		Best	652.0	1707.12	1403.26	1416.65
		Worst	5155.12	3522.21	4278.29	3306.7
CyberShake 1000	Makespan	Mean	1587.35	986.7	1061.56	1292.55
		Best	1414.63	864.5	863.05	1134.1
		Worst	1813.59	1113.64	1255.43	1443.65
	Cost	Mean	17114.44	3986.15	5324.06	8994.47
		Best	12157.46	2765.34	3576.07	7751.39
		Worst	24379.72	5473.88	7392.39	11287.91
Inspiral 100	Makespan	Mean	1481.07	1547.72	1481.35	1436.03
		Best	1101.57	1253.88	1241.37	1231.45
		Worst	2169.81	1868.23	1685.56	1684.66
	Cost	Mean	13305.37	14491.87	13220.95	12848.62
		Best	5074.79	7883.51	7356.71	8716.99
		Worst	28597.48	20225.57	19028.43	16853.28
Inspiral 1000	Makespan	Mean	19405.62	14207.17	15444.38	17018.94
		Best	17622.34	13117.62	13428.43	16046.52
		Worst	21944.4	16164.51	16993.87	19405.41
	Cost	Mean	239364.95	82853.11	110239.99	156874.23
		Best	190106.68	58321.82	73085.53	96068.68
		Worst	308933.12	130483.66	131098.06	202697.81

In case of second scenario, we have considered Google Traces 2019 tasks ranging from 100 to 1400. The nature of these workloads is independent i.e. no parent-child dependency exists. Figure 4 represents performance comparison of proposed IJPSO, IPSO, PSO and GWO algorithm using real world Google Traces 2019 workloads. The evaluation is considered on the basis of three vital metrics makespan, cost and throughput for 100 to 1400 tasks.

Table 4: Results Summary for Google Traces 2019 Workloads

Workload	Metric	Result Type	GWO	IJPSO	IPSO	PSO
Google Traces 100	Makespan	Mean	83.06	86.22	85.75	93.53
		Best	60.23	66.07	76.36	80.83
		Worst	108.35	116.36	93.14	107.62
	Cost	Mean	196.17	251.01	240.23	307.97
		Best	87.95	105.89	188.89	194.13
		Worst	462.91	444.32	313.28	415.62
Google Traces 200	Makespan	Mean	210.28	182.07	177.82	211.71
		Best	177.18	153.01	150.88	178.37
		Worst	275.78	219.26	200.22	259.85
	Cost	Mean	610.49	479.12	453.29	752.45
		Best	354.29	269.73	314.38	501.7
		Worst	1757.67	788.38	634.82	1349.45
Google Traces 400	Makespan	Mean	570.44	365.33	368.78	448.91

		Best	490.73	305.61	299.27	396.89
		Worst	690.99	445.6	462.38	523.18
	Cost	Mean	2418.05	839.61	919.74	1520.87
		Best	1536.8	495.91	496.23	983.48
		Worst	3845.06	1451.91	1687.24	2023.99
Google Traces 600	Makespan	Mean	960.64	530.65	542.69	697.01
		Best	841.53	404.08	496.73	633.85
		Worst	1087.11	845.59	727.73	810.23
	Cost	Mean	5175.6	1233.0	1251.15	2451.07
		Best	3422.88	491.34	900.89	1817.74
		Worst	7294.01	4434.24	2848.83	3430.13
Google Traces 800	Makespan	Mean	1353.18	683.74	710.82	943.81
		Best	1250.85	576.23	622.45	829.00
		Worst	1461.48	1026.13	838.95	1096.83
	Cost	Mean	8045.84	1408.43	1554.43	3298.15
		Best	6250.92	850.87	1030.81	2195.64
		Worst	10101.62	3975.01	2373.06	4545.19
Google Traces 1000	Makespan	Mean	1793.0	863.6	975.26	1181.84
		Best	1636.99	754.93	863.77	1015.12
		Worst	1966.4	1026.96	1125.54	1327.34
	Cost	Mean	11270.67	1597.09	2342.69	3944.65
		Best	8678.42	1087.63	1590.88	2479.98
		Worst	15139.69	2649.86	3537.54	5655.77
Google Traces 1200	Makespan	Mean	2136.45	1082.27	1134.44	1449.69
		Best	1993.81	807.86	1040.76	1341.1
		Worst	2269.32	1645.72	1245.23	1636.35
	Cost	Mean	14293.57	2459.53	2670.2	5217.55
		Best	11424.12	1000.99	2036.13	4080.67
		Worst	16838.34	7288.17	3661.39	7026.61
Google Traces 1400	Makespan	Mean	2672.94	1227.8	1291.23	1709.66
		Best	2349.4	1034.37	1092.12	1542.82
		Worst	4149.14	1516.12	1395.69	1891.95
	Cost	Mean	21955.28	2412.9	2808.71	6297.16
		Best	14336.41	1620.37	1833.19	4623.01
		Worst	88181.58	3773.35	3475.58	9002.05

As shown in Figure 4 (a), the makespan is getting increased for all algorithms with the increase of workloads. GWO displays the highest makespan. It is the indication of the poor performance at large scale workloads whereas the proposed IJPSO beats all algorithms. One thing is noticeable that the proposed IJPSO provides better performance at large scales. It shows the effective resources picking due to balancing in local and global searching as well as faster convergence. The JAYA operators help the IJPSO in local search refinement in Google Traces 2019 independents tasks also.

Figure 4 (b) depicts the cost utilization effects for all algorithms. As expected, the cost increases with the number of tasks because additional resources are required for computation. Again, GWO incurs higher cost compared to other algorithms and IJPSO uses minimum cost due to effective resource load balancing and picking of effective VMs due to adaptive inertia weight and JAYA local refinement operator.

Figure 4 (c) represents the throughput produced by proposed IJPSO, IPSO, PSO and GWO algorithms. For small workloads all algorithms produces almost comparable throughputs. But with the increment of workloads, the GWO performs worst and our proposed IJPSO algorithm outperforms other. The second best algorithm is IPSO for generating good throughput. The IJPSO is producing good results due to avoidance of poor solutions and maintaining the local and global searching balance due to JAYA algorithm operators and adaptive inertia weight.

Further the results summary of all algorithms regarding scientific workflows and Google Traces 2019 are also represented in Table 3 and 4 respectively.

The average improvement of proposed IJPSO over IPSO is 3.43% makespan and 13.84% cost in case of all CyberShake workflows. In case of Inspiral workflows, the average improvement of IJPSO over IPSO is 1.77%

in makespan and 7.62% in cost. For Google Traces 2019, the average improvement of proposed IJPSO is 3.12% in makespan and 9.79% in cost.
Compared to PSO, the average improvement in makespan of IJPSO is 3.60%, 4.37% and 21.54% for CyberShake, Inspiral, and Google Traces respectively. In case of cost, it is 14.83%, 17.20%, and 47.58% for CyberShake, Inspiral, and Google Traces respectively.

6. Conclusion and Future Directions

The emerging technologies are growing day by day and the cloud computing plays a vital role for providing online services on pay-per-basis. Task scheduling and load balancing are hot topics for research because these methods play a tremendous role to improve the performance of a cloud system. There are many meta-heuristic techniques available nowadays, but these are having some issues like weak exploitation and exploration which affect the performance of the system. In this paper, a novel hybrid algorithm named IJPSO has been designed to overcome the problem of local searching stagnation available in PSO algorithm. The proposed algorithm IJPSO has the capacity of good exploration and exploitation due to that it shows tremendous performance as compared to IPSO, PSO and GWO algorithms in terms of makespan, financial cost and throughput. In case of average makespan for all scientific workflows workloads, the IJPSO is around 1.77% to 3.43% better than its competitor IPSO. In case of average financial cost, the IJPSO is 7.62% to 13.84% better than IPSO. In case of Google Traces independent workloads, the IJPSO is 3.12 compared to IPSO in terms of makespan. In case of cost, the IJPSO is 9.79% better than IPSO. The better results are obtained due to the balance between exploration and exploitation, local refinement due to JAYA operators which cause good convergence and effective resource utilizations.

In future, this newly designed algorithm can be applied in various engineering fields and to achieve other objectives in cloud systems. The proposed algorithm can be applied at the hosts and datacenter levels. A new algorithm can also be developed based on this strategy.

References

1. Alouffi, B., Hasnain, M., Alharbi, A., Alosaimi, W., Alyami, H. and Ayaz, M. (2021). A Systematic Literature Review on Cloud Computing Security: Threats and Mitigation Strategies. IEEE Access, 9, 57792–57807.
2. Alzaqebah, A., Al-Sayyed, R. and Masadeh, R. (2019). Task Scheduling based on Modified Grey Wolf Optimizer in Cloud Computing Environment Abdullah. In Proceedings of the International Conference on New Trends in Computing Sciences, pp. 176–183.
3. Awad, A.I., El-Hefnawy, N.A. and Abdel-Kader, H.M. (2015). Enhanced Particle Swarm Optimization for Task Scheduling in Cloud Computing Environments. Procedia Computer Science, 65, 920–929.
4. Bhagwan, J. and Kumar, S. (2021). An H-CSO Algorithm for Workflow Scheduling in Heterogeneous Cloud Environment. International Journal of Intelligent Engineering & Systems, 14(5), 422–434.
5. Bhagwan, J. and Kumar, S. (2021). An HC-CSO Algorithm for Workflow Scheduling in Heterogeneous Cloud Computing System. International Journal of Advanced Computer Science and Applications, 12(6), 484–492.
6. Chen, S., Xu, Z., Tang, Y. and Liu, S. (2014). An Improved Particle Swarm Optimization Algorithm Based on Centroid and Exponential Inertia Weight. Mathematical Problems in Engineering, 2014, 1–14.
7. Chen, X., Cheng, L., Liu, C., Liu, Q. and Liu, J. (2019). A WOA-Based Optimization Approach for Task Scheduling in Cloud Computing Systems. IEEE Systems Journal, 14(3), 1–12.
8. Ebadifard, F. and Babamir, S.M. (2018). A PSO-Based Task Scheduling Algorithm Improved Using a Load-Balancing Technique for the Cloud Computing Environment. Concurrency and Computation: Practice and Experience, 30(12), 1–16.
9. Hemasian-Etefagh, F. and Safi-Esfahani, F. (2019). Dynamic Scheduling Applying New Population Grouping of Whales Meta-Heuristic in Cloud Computing. Soft Computing, 75(10).
10. Gohil, B.N. and Patel, D.R. (2018). An Improved Grey Wolf Optimizer (iGWO) for Load Balancing in Cloud Computing Environment. In Springer International Publishing, 11338.
11. Hussain Madni, S.H., Abd Latiff, M.S., Abdullahi, M., Abdulhamid, S.M. and Usman, M.J. (2017). Performance Comparison of Heuristic Algorithms for Task Scheduling in IaaS Cloud Computing Environment. PLOS ONE, 12(2), 1–26.
12. Jena, U.K., Das, P.K., Mishra, K. and Kabat, M.R. (2021). Improved Binary JAYA Algorithm-Based Scheduling Dynamic Cloud Requests for Cloud-Based Computing. In Proceedings of the International Conference on Computing, Communication and Networking Technologies, pp. 1–6.

13. Khurana, S. and Singh, R.K. (2019). Workflow Scheduling and Reliability Improvement by Hybrid Intelligence Optimization Approach with Task Ranking. EAI Endorsed Transactions on Scalable Information Systems, 7(24), 1–10.
14. Laabadi, S., Naimi, M., El Amri, H. and Achchab, B. (2020). A Binary Crow Search Algorithm for Solving Two-Dimensional Bin Packing Problem with Fixed Orientation. In Procedia Computer Science, 167, 809–818.
15. Luo, F., Yuan, Y., Ding, W. and Lu, H. (2018). An Improved Particle Swarm Optimization Algorithm Based on Adaptive Weight for Task Scheduling in Cloud Computing. In Proceedings of the 2nd International Conference on Computer Science and Application Engineering, 1282, 1–5.
16. Mansouri, N., Mohammad Hasani Zade, B. and Javidi, M.M. (2019). Hybrid Task Scheduling Strategy for Cloud Computing by Modified Particle Swarm Optimization and Fuzzy Theory. Computers & Industrial Engineering, 130, 597–633.
17. Mapetu, J.P.B., Chen, Z. and Kong, L. (2019). Low-Time Complexity and Low-Cost Binary Particle Swarm Optimization Algorithm for Task Scheduling and Load Balancing in Cloud Computing. Applied Intelligence, 49(9), 3308–3330.
18. Milan, S.T., Rajabion, L., Ranjbar, H. and Navimipour, N.J. (2019). Nature Inspired Meta-Heuristic Algorithms for Solving the Load-Balancing Problem in Cloud Environments. Computer & Operations Research, 110, 159–187.
19. Mishra, K., Pati, J. and Majhi, S.K. (2022). A Dynamic Load Scheduling in IaaS Cloud Using Binary JAYA Algorithm. Journal of King Saud University – Computer and Information Sciences, 34(8), 4914–4930.
20. Natesan, G. and Chokkalingam, A. (2019). Task Scheduling in Heterogeneous Cloud Environment Using Mean Grey Wolf Optimization Algorithm. ICT Express, 5(2), 110–114.
21. Natesan, G. and Chokkalingam, A. (2020). An Improved Grey Wolf Optimization Algorithm Based Task Scheduling in Cloud Computing Environment. The International Arab Journal of Information Technology, 17(1), 73–81.
22. Ngatman, M.F., Sharif, J.M. and Ngadi, M.A. (2017). A Study on Modified PSO Algorithm in Cloud Computing. In ICT International Student Project Conference, pp. 1–4.
23. Prasanna Kumar, K.R., Kousalya, K., Vishnuppriya, S., Ponni, S. and Logeswaran, K. (2021). Enhanced Crow Search Algorithm for Task Scheduling in Cloud Computing. In Materials Science and Engineering, 1055.
24. Pradhan, A., Bisoy, S.K. and Das, A. (2022). A Survey on PSO Based Meta-Heuristic Scheduling Mechanism in Cloud Computing Environment. Journal of King Saud University – Computer and Information Sciences, 34(8).
25. Saleh, H., Nashaat, H., Saber, W. and Harb, H.M. (2019). IPSO Task Scheduling Algorithm for Large Scale Data in Cloud Computing Environment. IEEE Access, 7, 5412–5420.
26. Singh, H., Tyagi, S. and Kumar, P. (2019). Crow Search Based Scheduling Algorithm for Load Balancing in Cloud Environment. International Journal of Innovative Technology and Exploring Engineering, 8(9), 1058–1064.
27. Ting, T.O., Shi, Y., Cheng, S. and Lee, S. (2012). Exponential Inertia Weight for Particle Swarm Optimization. In Lecture Notes in Computer Science, pp. 83–90.
28. Wu, D. (2018). Cloud Computing Task Scheduling Policy Based on Improved Particle Swarm Optimization. In Proceedings of the International Conference on Virtual Reality Intelligent Systems, pp. 99–101.
29. Xingjun, L., Zhiwei, S., Hongping, C. and Mohammed, B.O. (2020). A New Fuzzy-Based Method for Load Balancing in the Cloud-Based Internet of Things Using a Grey Wolf Optimization Algorithm. International Journal of Communication Systems, 33(8), 1–19.
30. Zhao, Y. (2024). Enhanced Butterfly Optimization Algorithm for Task Scheduling in Cloud Computing Environments. International Journal of Advanced Computer Science and Applications, 15(12), 435–443.