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

LBGWO: A Load Balancing Based Strategy for Large Scale Task Scheduling in Cloud Computing

Manoj[1], Jai Bhagwan[1*], Sandeep[2], Sanjeev Kumar[1], Ajay Kumar[3], Shivani Gupta[3]

[1]Department of Computer Science & Engineering, Guru Jambheshwar University of Science & Technology, Hisar, Haryana, India.
[2]Department of Artificial Intelligence & Data Science, Guru Jambheshwar University of Science & Technology, Hisar, Haryana, India.
[3]Department of Computer Science & Engineering, Indira Gandhi University, Meerpur, Haryana, India.
*Corresponding Author: drjaicse@gmail.com, ORCID: 0000-0002-8708-4029

Abstract
Efficient task scheduling and load balancing are challenging in heterogeneous cloud computing environments due to uneven workload distribution and increasing computational demands. This paper proposes LBGWO (Load-Balanced Grey Wolf Optimization) which integrates an adaptive load-balancing strategy with Grey Wolf Optimization. The proposed method finds overloaded VMs using an adaptive threshold based on mean load and standard deviation and reallocates tasks to the lest-loaded VMs. Experiments are conducted using Google Traces 2019 workloads of 2000-3500 tasks in CloudSim 3.0. The proposed LBGWO is compared with IPSO, PSO and GWO using makespan, degree of imbalance, response time, throughput and convergence. Results show that LBGWO algorithm consistently achieves superior performance with 13-25% improvement in makespan, 36-49% improvement in degree of imbalance, and 3-18% improvement in response time over its competitor IPSO algorithm. LBGWO also achieves the highest throughput and faster convergence, demonstrating its effectiveness for load-balanced cloud task scheduling.

Keywords – Cloud Computing, Degree of Imbalance, Makespan, LBGWO, Grey Wolf Optimization, Virtual Machines (VMs)

1 Introduction

Cloud Computing is a fundamental computing paradigm for delivering scalable, flexible and on-demand resources to end user via internet. The fastest growth of cloud-based applications requires efficient load distribution among heterogeneous virtual machines. In such heterogeneous environment, load balancing plays a crucial role in resource management concepts for as uneven load distribution can cause overloaded VMs, higher energy utilization and degrade Quality of Service (QoS). Consequently, an efficient load-balancing strategy should distribute tasks among available virtual machines while maintaining energy consumption, computation cost, throughput [1][4] etc. the load-balancing problem becomes crucial problem due to heterogeneous resources and dynamic workload. Furthermore, task-to-VM mapping is considered a combinatorial optimization problem which can make exhaustive search computationally impractical for large-scale cloud computing environment. Metaheuristic algorithms have received considerable attention due to the capacity of large space solution's exploration. Among the metaheuristic approaches, Particle Swarm Optimization has been widely used because of its simple structure and limited number of control parameters. It can rapidly identify promising solutions for NP-hard problems. Various PSO-based cloud task scheduling and load-balancing approaches have been developed so for to minimize makespan, reduce response time and distribute load uniformly among VMs [2][3][5][6]. Several scientists have enhanced conventional PSO to overcome premature convergence and insufficient exploration. Golchi *et al.* developed hybrid Firefly-Improve PSO method for cloud load balancing. This method demonstrated improvements in terms of average load, resource utilization and response time [1]. Mansouri *et al.* integrated modified PSO with fuzzy theory and used task, CPU, execution time, and RAM characteristics to improve scheduling and load distribution [2]. Wang *et al.* proposed an improved PSO using adaptive inertia weight and random factor correlation to achieve lower scheduling cost [3]. Pradhan and Bisoy designed LBMPSO especially for load balancing and the target of this research was to improve makespan and resource utilization [4]. Similarly, heuristic initialization strategies have been incorporated into PSO in order to improved convergence and reduce adverse effects of random initialization of population [5]. Other methods have been combined PSO with Q-learning, Firefly optimization, feedback control, and time-conscious scheduling in order to improve dynamic load management [6]-[9].

Another important swarm-intelligence based technique is Grey Wolf Optimization which models the leadership and hunting behaviour of grey wolves. It is also relatively having simple mathematical structure and this algorithm is also widely adopted for cloud task scheduling and load-balancing [11]-[20] Natesan and Chokkalingam designed a Mean GWO-based task scheduling method that achieved improved makespan and energy consumption compared to

conventional PSO and GWO [11]. Li *et al.* developed a fuzzy-based GWO algorithm for cloud-based IoT load balancing and focused on response time and degree of imbalance [12]. Arora and Dixit designed an Elephant Herd-GWO, a hybrid technique for reallocation of tasks from overloaded to under-loaded VMs [13]. In [14], a reliability-aware GWO has also been introduced to improve load balancing, response time and reducing operating cost [14]. Recent research increasingly integrated PSO and GWO algorithms to use their complementary search capacities. The Hybrid GWO-PSO methods improved global exploration, convergence speed and avoid local optima. For example, Janakraman and Priya proposed a hybrid GWO algorithm and Improved PSO strategy which used inertia weight method for cloud load balancing [18]. More recent work has investigated new GWO encoding strategies, dynamic archives and adaptive perturbation approaches to address the cloud workloads [19][20].

Although PSO and GWO have demonstrated considerable potential works for cloud load balancing. Still, challenges remain in achieving a robust balance between exploration and exploitation, avoiding premature convergence and uneven distribution of load. These limitations motivate the development of improved load distribution strategy for cloud workload scheduling in order to achieve overall system performance.

The major contributions of this research paper are discussed as:

1. Modelling of Makespan, degree of imbalance and response time.
2. Designed load balancing technique.
3. Integration of load balancing technique into GWO algorithm.
4. Task priorities for Google Traces 2019 dataset have been used based on HEFT policy across all algorithms considered in this research, ensuring a fair comparison.
5. Comparison of designed LBGWO algorithm with IPSO, GWO and PSO.

Further, the paper is structured in five sections. Section 2 discusses literature review, system modeling and problem definition in described in section 3, section 4 explains proposed method, section 5 explains simulation results and finally conclusion and future work is describe in section 6.

2 Literature Review

This section covers the recent researches investigated using two most widely used algorithms named PSO and GWO. Research on swarm intelligence-based load balancing has gained a potential evolution from conventional PSO and GWO methods toward adaptive, hybrid and multi-objective optimization methods. Early PSO based researches focused on improve task scheduling to reduce execution time. Golchi *et al.* designed a Firefly based Improved PSO algorithm for cloud load balancing. Firefly optimization method was used to improve the initial search population and IPSO was designed to optimize the task allocation process. The results noted improvements in average load, response time, throughput and makespan [1]. Mansouri *et al.* proposed FMPSO algorithm which is a combination of modified PSO with fuzzy inference system to improve cloud load balancing and throughput. The method used task length, CPU speed, RAM size and execution time into decision process and displayed improvements in degree of imbalance and makespan [2]. Wang *et al.* introduced an improved PSO method based on inertia weight and random factor correlation for task mapping. This approach reduced the computation cost compared to sequential, greedy, conventional PSO and PSO methods [3]. Pradhan and Bisoy designed LBMPSO algorithm for load balancing. It used fitness function based on VM execution time. The CloudSim based experiments proved reduced makespan and improved resource utilization [4]. Alsaidy *et al.* focused on the weakness of random PSO initialization by suing LJFP and MCT heuristic methods for initialization of swarm. The LJFP-PSO and MCT-PSO methods improved makespan, degree of imbalance, and energy consumption compared to PSO [5]. Jena *et al.* introduced QMPSO algorithm which is a hybridization of MPSO with improved Q-learning for dynamic load balancing. This method optimized VM waiting time, workload distribution and throughput [6]. Devaraj *et al.* introduced FIMPSO algorithm by incorporating Firefly optimization in MOPSO for energy efficient load-balancing. The proposed method showed improved results in terms of resource utilization, response time while maintaining the cloud load [7]. Ghafir *et al.* later designed an intelligent PSO-based feedback controller for considering task scheduling, resource allocation and VMs migration. The multi-objective strategy considered throughput, scalability, response time and migration related issues [8]. Menaka and Kumar introduced SPSO-TC which is a combination of Supportive PSO and Time-Conscious scheduling methods to consider makespan and load distribution while minimizing energy [9]. Arora and Banyal further designed the combination of PSO and GWO methods. The PSO-GWO methods showed improvements in terms of execution cost and time compared to the individual algorithms [10].

GWO-based research has also followed a similar progression. Natesan and Chokkalingam introduced Mean-GWO method for cloud task scheduling. The results reported improvements in makespan and energy consumption over PSO and GWO [11]. Li *et al.* developed a fuzzy theory and GWO based load-balancing strategy for cloud-based IoT environment. The designed algorithm targeted response time and load-imbalance reduction [12]. Arora and Dixit

proposed EHGWO method which an integration of Elephant Herd Optimization and GWO approaches to reallocate tasks from overloaded to under-loaded VMs. This method considered VM load, capacity, migration cost and makespan for research metrics [13]. Sefati *et al.* designed a reliability-based GWO load-balancing method and implemented it using CloudSim. The method demonstrated efficiency in cost and response time reduction while maintaining reliable resource allocation [14]. Gohil and Patel introduced an Improved GWO method particularly for cloud load-balancing. After comparison with Harmony Search, ABC, PSO and conventional GWO, it reported improved resource utilization and system performance [15]. Abed-alguni and Alawad designed a Distributed GWO for workflow scheduling which considered computation and data-transmission costs. This proposed method showed competitive performance against PSO, GWO and Binary PSO [16]. Alsadie proposed TSMGWO algorithm which is a multi-objective algorithm for cloud datacenter. The algorithm considered makespan, throughput and resource utilization as performance metrics [17]. Recent studies emphasize adaptive and hybridization strategies. Janakiraman and Priya proposed integrated GWO and improved PSO, chaos, and multidimensional learning. This method improved exploration-exploitation and reduced degree of imbalance, latency and makespan [18]. Huang *et al.* introduced GWO with a new one-dimensional encoding mechanism and greedy replacement for cloud task scheduling. It improved the scheduling performance and reduced the effective search-space dimensionality [19]. Aburinya *et al.* introduced an enhanced dynamic GWO algorithm using dynamic elite archive and stochastic levy flight based strategy to main the diversity under dynamic workloads. The CloudSim experiments proved the efficiency of this newly designed algorithm over GWO, PSO, GA and ABC in load balancing and resource utilization [20].

The literature indicates that standard PSO and GWO provide effective foundations for cloud load balancing. But their performance may be compromised by premature convergence, inadequate diversity, difficulty in load distribution etc. So an improved load balancing algorithm is required.

3 System Model and Problem Definition

This section is discussing problem formulation and system modelling.

3.1 Problem Formulation

This research considers one datacenter, one host and 80 heterogeneous VMs for simulation. Let $T = \{T_1, T_2, T_3, T_4, \dots, T_n\}$ be the set of n tasks and $VM = \{VM_1, VM_2, VM_3, VM_4, \dots, VM_m\}$ be the set of m virtual machines. The scheduler should determine a mapping $T \rightarrow VM$ such that each task is assigned VM. Let's consider an example of 18 tasks and 4 VMs mapping which can be seen in Table 1.

Table 1: Task-VM Mapping Representation

VM1	T1	T3	T7	T10	T15	-
VM2	T4	T5	T9	T16	-	-
VM3	T2	T6	T8	T12	T13	T14
VM4	T11	T17	T18	-	-	-

3.2 Makespan Model

In cloud computing environment, the total completion time taken by all tasks in a set of tasks T is called as makespan [23]. Its mathematical modelling is represented in Eq. (1).

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

Where, $i \in T$, T is set of all tasks and F_i is finish time of tasks i.

3.3 Degree of Imbalance

The degree of imbalance (DI) tells the load distribution among VMs. The DI is calculated using Eq. (2).

$$DI = (T_{max} - T_{min})/T_{avg} \tag{2}$$

Where, T_{max}, T_{min} and T_{avg} define maximum, minimum and average loads among all VMs.

3.4 Response Time

The total time taken by a cloud system to respond to a user's request is called as response time, starting from the moment the task is submitted until the result is returned to the user. The response time is calculated using Eq. (3).

$$RT_i = F_i - S_i \tag{3}$$

Where, F_i is completion time and S_i is task submission or arrival time.

3.5 Throughput

The throughput [23] metric is used to observe the system's performance. It is computed using Eq. (4). We have not included the throughput in fitness function to avoid a complex fitness function.

$$\text{throughput} = \text{total tasks}/\text{makespan} \tag{4}$$

3.6 Fitness Function

The fitness function used in this research considered to minimize makespan, response time and degree of imbalance (DI). For optimization, we have combined all these metrics in a single objective function (F_x) which is represented in Eq. (5). Number of DC and VMs are used here to normalize the fitness values.

$$F_x = \frac{w_1 \times \text{makespan} + w_2 \times \text{DI} + w_3 \times \text{Response Time}}{\text{DC} \times \text{VMs}} \tag{5}$$

Where, w_1, w_2 and w_3 are the weights used to give the priorities to makespan, DI and Response Time. Here, a higher priority is given to makespan to stable the system performance; the second higher priority is set for DI for considering load balancing. The values of w_1, w_2 and w_3 are 0.5, 0.3 and 0.2 respectively given that $w_1 + w_2 + w_3 = 1$.

4 Methods and Workload Data

This section discusses the proposed method and dataset used in this research for experiments.

4.1 Dataset

In this research, we have real word dataset i.e. Google Traces 2019 dataset. We have used 2000-3500 tasks having random lengths from 5000-416851 MI based CPU and Run Time requirements. The missing values are adjusted within 5000-15000 MI range, the same is discussed in [24].

4.2 Priority Algorithm

The priority algorithm is designed based on HEFT [21]-[24] algorithm which can work for workflows as well as independent tasks (synthetic or real work dataset like Google Traces 2019).

Algorithm 1: Priority Algorithm (Based on HEFT)

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)
```

The priority algorithm is used for deciding task priorities for initial population generation for all algorithms used in this research for fair comparison. The steps of this priority algorithm are given in Algorithm 1. In case of synthetic or other types of independent tasks, the tasks' parent-child dependency is considered nil.

4.3 GWO Theory

Grey Wolf Optimization algorithm [25] is a population based swarm-intelligence algorithm inspired by the social hierarchy and hunting behaviour of grey wolves. In GWO, solutions are ranked as alpha (α), beta (β), delta (δ) and omega (ω). The α, β, and δ wolves guide the remaining wolves toward the optimum solution. The mathematical modelling of GWO is discussed below.

1. Encircling Prey

The position of a wolf is updated based on its distance from the prey:

$$D = |C \times X_p - X| \quad (6)$$

$$X(t+1) = X - A \times D \quad (7)$$

Similarly,

$$A = 2ar_1 - a \quad (8)$$

$$C = 2r_2 \quad (9)$$

Here, X_p is the prey position, X is the current wolf position, and r_1 and r_2 are random variable in (0, 1).

The parameter a decreases linearly from 2 to 0:

$$a = 2 - \frac{2t}{T} \quad (10)$$

Where, t is the current iteration and T is the maximum number of iterations.

2. Hunting

The positions of α, β, and δ wolves are used to guide each wolf:

$$D_\alpha = |C_1 X_\alpha - X| \quad (11)$$

$$X_1 = X_\alpha - A_1 D_\alpha \quad (12)$$

Similarly,

$$X_2 = X_\beta - A_2 D_\beta \quad (13)$$

$$X_3 = X_\delta - A_3 D_\delta \quad (14)$$

The final position is:

$$X(t+1) = \frac{X_1 + X_2 + X_3}{3} \quad (15)$$

4.4 Load Balancing Strategy

In this research, an adaptive load balancing strategy is introduced. Let L_v is the workload or load of VM v. It is calculated using Eq. (16).

$$L_v = \sum_{t=1}^{T} ET_{t,v} I(x_t = v) \quad (16)$$

Where, $ET_{t,v}$ is the execution time of task t on VM v and x_t is the VM assigned to task t.

The average VM load is:

$$L_{avg} = \frac{1}{V} \sum_{v=1}^{V} L_v \quad (17)$$

The load variance is:

$$\sigma^2 = \frac{1}{V} \sum_{v=1}^{V} (L_v - L_{avg})^2 \quad (18)$$

and standard deviation is:

$$\sigma = \sqrt{\frac{1}{V} \sum_{v=1}^{V} (L_v - L_{avg})^2} \quad (19)$$

The adaptive overload threshold is calculated using Eq. (20).

$$L_{th} = L_{avg} + \sigma \quad (20)$$

Therefore, VM v is considered overloaded when:

$$L_v > L_{th} \quad (21)$$

For an overloaded VM, the algorithm selects the least-loaded VM using Eq. (22).

$$v^* = \arg min_v L_v \quad (22)$$

If the task is moved VM v to VM v^*, the loads are updated as:

$$L'_v = L_v - ET_{t,v} \quad (23)$$

$$L'_{v^*} = L_{v^*} - ET_{t,v^*} \quad (24)$$

and task assignment becomes:

$$x_t = v^* \quad (25)$$

The task shifting stops for the current overloaded VM when:

$$L'_v \leq L_{th} \quad (26)$$

The introduced load balancing strategy has been presented in Algorithm 2 for better understanding.

Algorithm 2: Proposed Load Balancing Strategy

Input: Solution sol
Output: Rescheduled solution

```
1.  loads ← CalculateVMLoads (sol)
2.  avg ← Mean(loads)
3.  Calculate Standard Deviation: stdDev (using Eq. 19)
4.  threshold ← avg + stdDev
5.  For each t in sol Do
6.  |  vm ← current VM assigned to task t
7.  |  If loads[vm]>threshold Then
8.  |  |  target ← leastLoadedVM(loads)
9.  |  |  If target ≠ vm Then
10. |  |  |  Update loads using ET[t][vm] and ET[t][target]
11. |  |  |  Move task t from vm to target
12. |  |  |  If loads[vm]≤threshold Then
13. |  |  |  |  Break
14. |  |  |  End If
15. |  |  End If
16. |  End If
17. End For
18. return sol
```

4.5 Proposed LBGWO Algorithm

The introduced load balancing strategy has been integrated in GWO algorithm which can be understood using Algorithm 3.

Algorithm 3: Proposed LBGWO Algorithm

Input: Generate population pop by Priority Algorithm 1, max_iterations etc.
Output: Best Schedule gBest

```
1.  For iter = 0 to max_iterations Do
2.  |  Evaluate fitness of all wolves
3.  |  Find α, β and δ wolves in pop
4.  |  Calculate control parameter a ← 2 − 2 × iter/max_iterations
5.  |  For each wolf x in pop Do
6.  |  |  For each dimension d Do
7.  |  |  |  Calculate A1, C1, Dα and x1 using α wolf
8.  |  |  |  Calculate A2, C2, Dβ and x2 using β wolf
9.  |  |  |  Calculate A3, C3, Dδ and x3 using δ wolf
10. |  |  |  x[d] ← (x1 + x2 + x3)/3
11. |  |  End For
12. |  |  Call Algorithm 2
13. |  End For
14. End For
15. Return Best Schedule gBest
```

In Algorithm 3, first VMs are initialized using Algorithm 1. Next, after evaluation of fitness using Eq. (5) the control parameter is calculated for exploration and exploitation. The position update takes place (see line 10.) in next step. The Algorithm 2 (in line 12) finds out the overloaded VM and shift the load to under-loaded VM. This process continues till the maximum number of iterations is achieved. Finally, the best scheduling or Task-VM mapping returned.

5 Results and Discussion

The simulation is performed using CloudSim 3.0 simulator which is written in Java programming language. In this research, all algorithms have been implemented for a fair comparison 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 Pro 64-bit OS. We have created 3 types of 80 heterogeneous VMs containing computing power between 500-1500 MIPS, 512-1536 MB RAM and 1000 Mbps bandwidth. We have tested all algorithms for 15 rounds to check the stability and fair performance. The other simulation parameters have been depicted in Table 1. The performance parameters considered in this research are makespan, DI (degree of imbalance), response time and throughput as described earlier. The number of population is set to 50 and maximum iterations are 200 for all algorithms used in this research. The workload used in this research is Google Traces 2019 dataset containing 2000-3500 tasks. Table 2 represents the summary of results obtained after 15 runs of all algorithms.

Table 2: Summary of Makespan, DI and Response Time

Workload	Metric	Result Type	IPSO	LBGWO	PSO	GWO
GoogleTrace-2000	makespan	Mean	1852.538	1434.004	4908.483	4585.321
	makespan	Best	1050.227	1094.432	3524.837	4260.783
	makespan	Worst	2252.845	1927.365	5406.020	4846.273
	DI	Mean	8.535	4.330	69.806	20.808
	DI	Best	4.521	3.397	28.454	17.254
	DI	Worst	10.512	5.916	79.624	24.627
	Response Time	Mean	865.711	742.960	2475.316	2266.866
	Response Time	Best	508.971	332.251	1791.931	2050.777
	Response Time	Worst	1127.065	1259.296	2721.558	2405.101
GoogleTrace-2300	makespan	Mean	1739.561	1504.108	4821.486	4571.338
	makespan	Best	1104.068	1141.800	3366.622	4271.190
	makespan	Worst	2236.365	2126.658	5178.692	4921.912
	DI	Mean	7.855	4.950	67.339	21.552
	DI	Best	4.798	4.029	27.770	17.295
	DI	Worst	11.289	6.328	79.390	25.491
	Response Time	Mean	763.320	793.354	2425.040	2246.675
	Response Time	Best	527.612	545.932	1595.401	2100.307
	Response Time	Worst	1079.762	1098.753	2639.854	2381.850
GoogleTrace-2600	makespan	Mean	1903.828	1414.230	4765.163	4790.902
	makespan	Best	1351.218	1083.388	3135.087	4362.545
	makespan	Worst	2475.240	1860.322	5222.917	5358.472
	DI	Mean	8.535	4.654	65.966	22.606
	DI	Best	5.252	2.831	27.890	19.237
	DI	Worst	12.566	6.497	79.080	27.087
	Response Time	Mean	897.537	728.112	2404.227	2398.345
	Response Time	Best	589.381	437.588	1588.020	2195.314
	Response Time	Worst	1238.227	1026.237	2684.274	2634.315
GoogleTrace-2900	makespan	Mean	1862.475	1471.127	4741.476	4680.621
	makespan	Best	1070.678	931.428	3300.382	4448.787
	makespan	Worst	2217.952	2114.353	5335.648	4918.440
	DI	Mean	8.049	4.736	64.720	23.031
	DI	Best	4.477	3.071	26.631	20.262
	DI	Worst	12.452	7.267	78.796	26.339
	Response Time	Mean	852.487	791.440	2382.103	2318.347
	Response Time	Best	502.851	316.449	1667.298	2233.849
	Response Time	Worst	963.107	1266.687	2709.482	2478.892
GoogleTrace-3200	makespan	Mean	1666.018	1346.065	4559.731	4724.981
	makespan	Best	1031.428	1033.745	3365.193	4240.432
	makespan	Worst	2238.455	1809.810	5251.790	5086.362
	DI	Mean	7.282	4.339	59.317	22.225
	DI	Best	4.666	3.128	24.619	20.504
	DI	Worst	11.913	6.394	78.717	24.307
	Response Time	Mean	780.234	683.603	2284.695	2334.871
	Response Time	Best	526.517	457.377	1673.681	2090.712
	Response Time	Worst	1103.292	946.667	2640.487	2508.927
GoogleTrace-3500	makespan	Mean	1830.179	1374.404	4880.448	4677.870
	makespan	Best	1180.047	905.672	3283.703	4281.697
	makespan	Worst	2467.923	2263.052	5211.672	5126.843
	DI	Mean	8.201	4.514	68.070	21.540
	DI	Best	4.434	2.662	27.021	19.318
	DI	Worst	12.185	5.916	79.496	24.414
	Response Time	Mean	839.390	691.748	2464.475	2306.626
	Response Time	Best	544.307	370.326	1615.486	2049.641
	Response Time	Worst	1062.325	1401.549	2654.033	2509.987

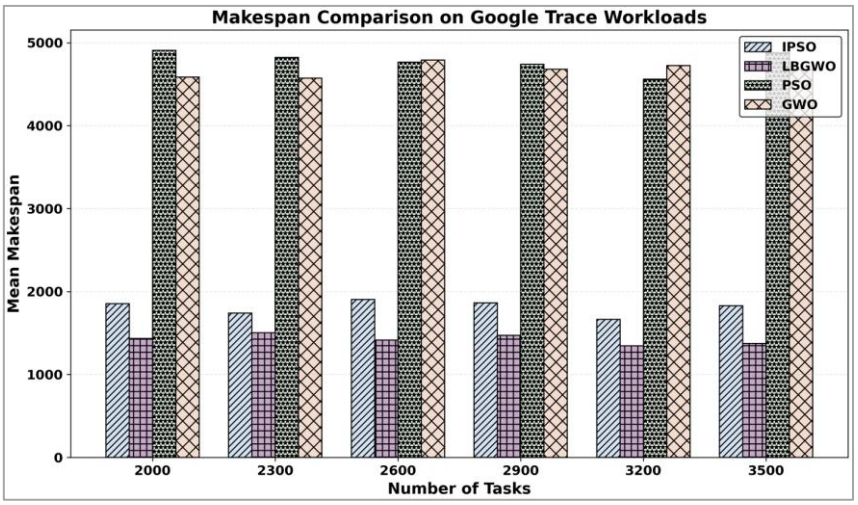

Figure 1: Makespan Comparison

Figure 1 compares the mean makespan of proposed LBGWO, IPSO, GWO and PSO algorithms for Google Traces 2019 workloads ranging from 2000 to 3500 tasks. LBGWO algorithm achives the minimum makespan compared to other algorithms, indicating better scheduling efficiency. IPSO performs second best and PSO performs the poor efficiency. LBGWO algorithm remains stable as the number of tasks increases, demonstrating good scability and load balancing capacity. The improvement of LBGWO over IPSO is recorded 13% - 25% in terms of makespan. This is due to the excellent load balancing stratgy and exploration capacity of LBGWO algorithm.

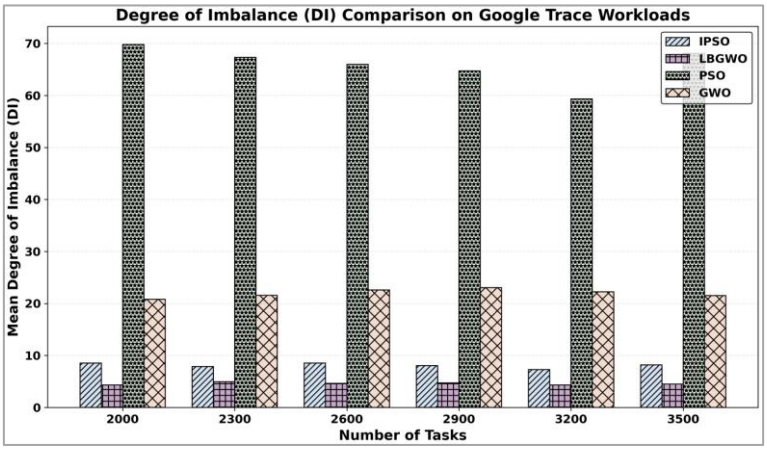

Figure 2: DI Comparison

Figure 2 demonstrates the results of degree of imbalance achieved during scheduling of 2000 – 3500 tasks for all algorithms. The proposed LBGWO algorithm distributed the load among VMs efficiently compared to IPSO, GWO and PSO algorithms. PSO algorithm is most imbalanced algorithm found in this research. IPSO algorithm is performing better than all algorithms except the proposed LBGWO. The load balancing strategy proposed in this research helps the LBGWO algorithm to distribute the load in a better way, as a result the proposed LBGWO method achieved improvement of 36 – 49% in account of degree of imbalance compared to IPSO algorithm.

Figure 3 compares the response time of IPSO, PSO, GWO and proposed LBGWO for 2000 – 3500 tasks taken from Google Traces 2019. LBGWO algorithm consistently achieves the lowest response time, indicating fast processing. IPSO algorithm has slightly higher response time than proposed LBGWO algorithm whereas PSO shows higher response time. The improved achieved by LBGWO over IPSO is 3% – 18% in terms of response time because our proposed load balancing strategy repairs the schedule efficiently.

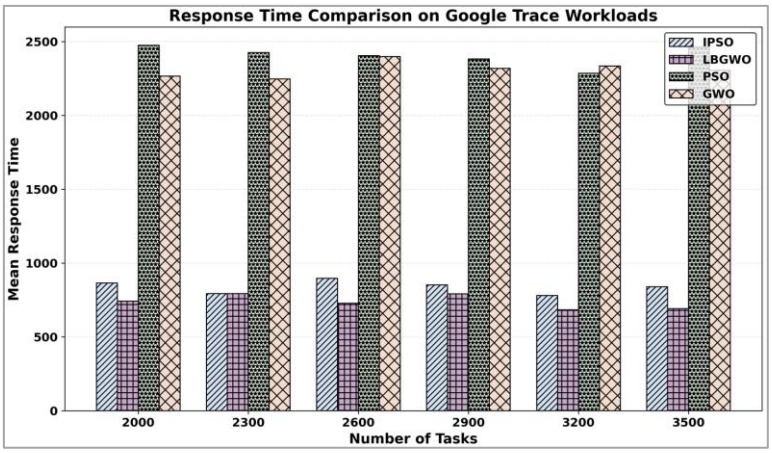

Figure 3: Response Time Comparison

Figure 4 demonstrates the comparison of throughput of proposed LBGWO, GWO, IPSO and PSO algorithms for 2000 – 3500 tasks. The graph clearly indicates that proposed LBGWO achieves the highest throughput across all tasks. The IPSO is second best algorithm as it beats other algorithm except proposed LBGWO algorithm. With increasing tasks, proposed LBGWO increases its throughput that means it can process more tasks in a given time. The superior throughput of LBGWO algorithm is consistent with its lower makespan, response time and degree of imbalance, suggesting more efficient use of available resources.

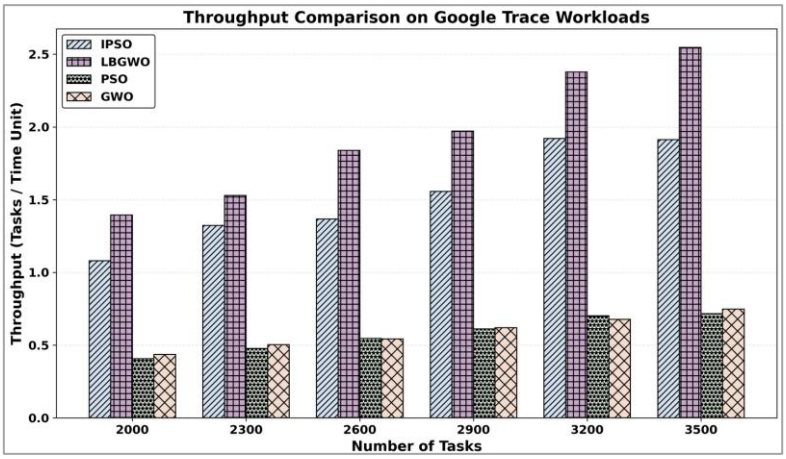

Figure 4: Throughput Comparison

Figure 5 shows convergence behaviour of LBGWO, IPSO, PSO and GWO algorithms for real word dataset Google Traces 2019 containing 2000 – 3500 tasks. The LBGWO converges fastest and achieves the lowest fitness in all six workloads. It fitness drops sharply during early iterations and then gradually improves. It is indicating exploitation of promising solutions. IPSO also converges rapidly at the beginning and reaches a relatively low fitness but its final fitness is higher than LBGWO. PSO shows slower convergence and stabilizes at a much higher fitness value. GWO initially decreases gradually and requires more iteration to converge. As the workload increases from 2000 to 3500 tasks, LBGWO maintains a similar convergence pattern and remains the best performing algorithm.

Overall, the convergence curves demonstrate that LBGWO method provides both faster convergence and better solution quality and it is consistently reaching lowest fitness across all Google Traces workloads. This indicates that the proposed load-balancing repair mechanism enhances the standard GWO search process by helping the algorithm better solution quality and more balanced solutions.

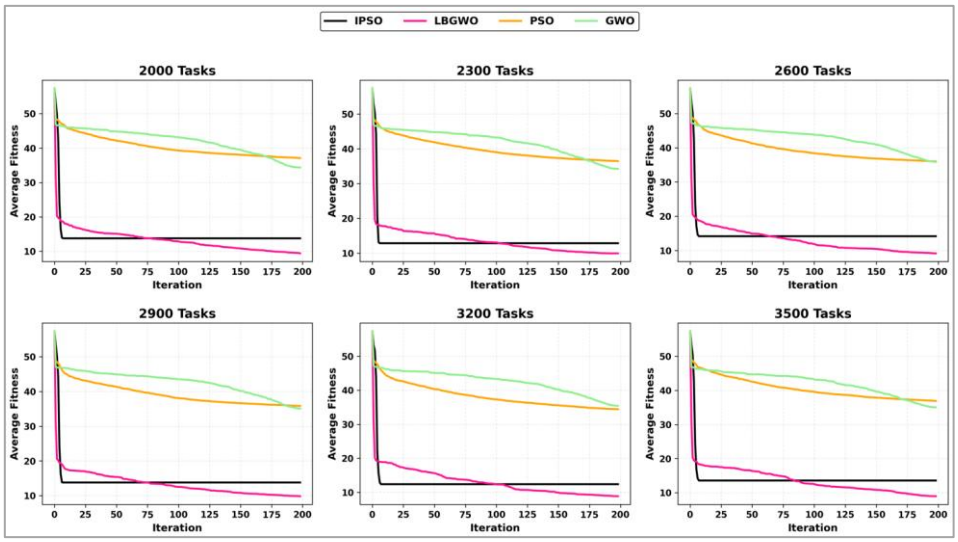

Figure 5: Convergence Results of All Algorithms

Table 3 is representing the percentage of improvement of proposed LBGWO over its competitor IPSO algorithm.

Table 3: Percentage of Improvement of LBGWO over IPSO

Workload	Makespan (%)	DI (%)	Response Time (%)
GT - 2000	22.59	49.27	14.18
GT - 2300	13.54	36.98	3.79
GT - 2600	25.72	45.47	18.88
GT - 2900	21.01	41.16	7.16
GT - 3200	19.20	40.41	12.38
GT - 3500	24.90	44.96	17.59

6 Conclusions and Future Direction

This paper proposed LBGWO, an improved GWO-based task scheduling algorithm incorporating an adaptive load-balancing strategy. The proposed method effectively identifies overloaded VMs and reallocates tasks to under-loaded VMs. Experimental evaluation on Google Traces 2019 workloads using CloudSim 3.0 demonstrates that LBGWO consistently outperforms IPSO, PSO, and GWO in terms of makespan, degree of imbalance, response time, and throughput. The convergence results further show faster convergence and better solution quality. Overall, the results confirm that integrating adaptive load balancing with GWO provides an effective approach for improving workload distribution and scheduling performance in heterogeneous cloud environments.

In future, the proposed algorithm can be test for more datasets and workflows. The simulation environments can be enlarged i.e. number of datacenter or host can be increased. More efficient swarm intelligence algorithm can also be designed to solve NP-hard scheduling problem.

References

1. M. M. Golchi, S. Saraeian, and M. Heydari, “A hybrid of firefly and improved particle swarm optimization algorithms for load balancing in cloud environments: Performance evaluation,” Computer Networks, vol. 162, Art. no. 106860, 2019, doi: 10.1016/j.comnet.2019.106860.
2. N. Mansouri, B. M. Hasani Zade, and M. M. Javidi, “Hybrid task scheduling strategy for cloud computing by modified particle swarm optimization and fuzzy theory,” Computers & Industrial Engineering, vol. 130, pp. 597–633, 2019, doi: 10.1016/j.cie.2019.03.006.
3. 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.
4. A. Pradhan and S. K. Bisoy, “A novel load balancing technique for cloud computing platform based on PSO,” Journal of King Saud University—Computer and Information Sciences, vol. 34, no. 7, pp. 3988–3995, 2022, doi: 10.1016/j.jksuci.2020.10.016.
5. S. A. Alsaidy, A. D. Abbood, and M. A. Sahib, “Heuristic initialization of PSO task scheduling algorithm in cloud computing,” Journal of King Saud University—Computer and Information Sciences, vol. 34, no. 6, pp. 2370–2382, 2022, doi: 10.1016/j.jksuci.2020.11.002.

6. U. K. Jena, P. K. Das, and M. R. Kabat, "Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment," Journal of King Saud University—Computer and Information Sciences, vol. 34, no. 6, pp. 2332–2342, 2022, doi: 10.1016/j.jksuci.2020.01.012.
7. A. F. S. Devaraj, M. Elhoseny, S. Dhanasekaran, E. L. Lydia, and K. Shankar, "Hybridization of firefly and improved multi-objective particle swarm optimization algorithm for energy efficient load balancing in cloud computing environments," Journal of Parallel and Distributed Computing, vol. 142, pp. 36–45, 2020, doi: 10.1016/j.jpdc.2020.03.022.
8. S. Ghafir, M. A. Alam, F. Siddiqui, and S. Naaz, "Load balancing in cloud computing via intelligent PSO-based feedback controller," Sustainable Computing: Informatics and Systems, vol. 41, Art. no. 100948, 2024, doi: 10.1016/j.suscom.2023.100948.
9. M. Menaka and K. S. Sendhil Kumar, "Supportive particle swarm optimization with time-conscious scheduling (SPSO-TCS) algorithm in cloud computing for optimized load balancing," International Journal of Cognitive Computing in Engineering, vol. 5, pp. 192–198, 2024, doi: 10.1016/j.ijcce.2024.05.002.
10. N. Arora and R. K. Banyal, "Workflow scheduling using particle swarm optimization and gray wolf optimization algorithm in cloud computing," Concurrency and Computation: Practice and Experience, vol. 33, Art. no. e6281, 2021, doi: 10.1002/cpe.6281.
11. G. Natesan and A. Chokkalingam, "Optimal task scheduling in the cloud environment using a mean Grey Wolf Optimization algorithm," International Journal of Technology, vol. 10, no. 1, pp. 126–136, 2019, doi: 10.14716/ijtech.v10i1.1972.
12. X. Li, Z. Shao, H. Cheng, and B. O. Mohammed, "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, vol. 33, no. 8, Art. no. e4370, 2020, doi: 10.1002/dac.4370.
13. P. Arora and A. Dixit, "An elephant herd Grey Wolf Optimization (EHGWO) algorithm for load balancing in cloud," International Journal of Pervasive Computing and Communications, vol. 16, no. 3, pp. 259–277, 2020, doi: 10.1108/IJPCC-09-2019-0070.
14. S. Sefati, M. Mousavinasab, and R. Zareh Farkhady, "Load balancing in cloud computing environment using the Grey Wolf Optimization algorithm based on the reliability: Performance evaluation," The Journal of Supercomputing, vol. 78, no. 1, pp. 18–42, 2022, doi: 10.1007/s11227-021-03810-8.
15. B. N. Gohil and D. R. Patel, "Load balancing in cloud using improved gray wolf optimizer," Concurrency and Computation: Practice and Experience, vol. 34, no. 11, Art. no. e6888, 2022, doi: 10.1002/cpe.6888.
16. B. H. Abed-alguni and N. A. Alawad, "Distributed Grey Wolf Optimizer for scheduling of workflow applications in cloud environments," Applied Soft Computing, vol. 102, Art. no. 107113, 2021, doi: 10.1016/j.asoc.2021.107113.
17. D. Alsadie, "TSMGWO: Optimizing task schedule using multi-objectives Grey Wolf Optimizer for cloud data centers," IEEE Access, vol. 9, pp. 37707–37725, 2021, doi: 10.1109/ACCESS.2021.3063723.
18. S. Janakiraman and D. P. M., "Hybrid grey wolf and improved particle swarm optimization with adaptive inertial weight-based multi-dimensional learning strategy for load balancing in cloud environments," Sustainable Computing: Informatics and Systems, vol. 38, Art. no. 100875, 2023, doi: 10.1016/j.suscom.2023.100875.
19. X. Huang, M. Xie, D. An, S. Su, and Z. Zhang, "Task scheduling in cloud computing based on grey wolf optimization with a new encoding mechanism," Parallel Computing, vol. 122, Art. no. 103111, 2024, doi: 10.1016/j.parco.2024.103111.
20. B. B. Aburinya, M. I. Daabo, and C. I. Nakpih, "A proposed enhanced dynamic grey wolf optimization algorithm for cloud load balancing," Journal of Electrical Systems and Information Technology, vol. 13, Art. no. 27, 2026, doi: 10.1186/s43067-026-00331-3.
21. 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.
22. J. Bhagwan, and S. Kumar, "Independent task scheduling in cloud computing using meta-heuristic HC-CSO algorithm," International Journal of Advanced Computer Science and Applications, vol. 12, no. 7, pp. 207-214, 2021, doi: 10.14569/IJACSA.2021.0120724.
23. J. Bhagwan, S. Rani, Manoj, S. Godara, Yashavi, 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. 3s, 2026, doi: 10.51483/IJAIML.6.2s.2026.399-412.
24. J. Bhagwan, S. Rani, Sanjeev Kumar, and S. Godara, "DWCSO: a hybrid intelligent algorithm for multi-type workload scheduling in cloud computing," International Journal Computer Information Systems and Industrial Management Applications, vol. 10, no. 10s, pp. 227-244, 2026, doi: 10.70917/ijcisim-2026-3580.
25. S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimization," Advances in Engineering Software, vol. 69, pp. 46-61, 2014, doi: 10.1016/j.advengsoft.2013.12.007.