Int. J. Nonlinear Anal. Appl. 10 (2019) No. 2, 177-188
ISSN: 2008-6822 (electronic)
https://dx.doi.org/10.22075/IJNAA.2019.4184

Workflow Scheduling in Cloud Computing Environment using Hybrid CSO-DA

A. Pourghaffari, M. Barari

Malek Ashtar University of Technology, Tehran, Iran

Abstract

With advances in virtualization technology, cloud computing has become the most powerful and promising platform for business, academia, public and government organizations. Scheduling these workflows and load balancing to get better success rate becomes a challenging issue in cloud computing. In this paper, we used Cats and Dragonfly Optimization (CSO-DA) algorithm to balance the Load in the process of allocating resources to virtual machines in cloud computing in order to improve the speed and accuracy of scheduling. The proposed method consists of the following steps: initialization of the algorithm and cloud computing, determining the number of virtual machines and the number of tasks, implementing a dragonfly optimization algorithm for choosing the best host and implementing a cat collapse algorithm for balancing the load and Schedule tasks between virtual machines. Our experiments show that as far as run time, response time, task immigration and significant load balances are concerned, our proposed model combining cat and dragonfly optimization algorithms achieved better performance in allocating resources and load balance between virtual machines than other methods.

Keywords: Task Scheduling, Load Balance, Cat Optimization Algorithm, Dragonfly Optimization, Cloud Computing.
2010 MSC: 68T20.

1. Introduction

Cloud computing can be described as the ability to share processing resources among users. Recent developments in virtualization have resulted in its proliferation across data centers [1]. User programs that run independently of hardware infrastructure and virtual machines (VM) migrate from one host to another, needless stopping of them is required for the development of data centers [2]. Cloud computing is distributed computing on the internet that makes computing resources

*A. Pourghaffari
Email addresses: pourghaffari@mut.ac.ir (A. Pourghaffari), barari@ict.gov.ir (M. Barari)

Received: December 2017 *Revised:* November 2018

available for users in a sharing model [3, 4]. Resource allocation, load balancing, energy consumption management, and task scheduling improvements are important challenges to cloud computing [5]. Data centers should schedule tasks dynamically to avoid having busy or idle hosts and balance the load between them [6, 7]. Response time reduction, fault tolerance improvement, better scalability, increasing user satisfaction, reducing heat and greenhouse gases through optimization of electrical energy consumption are the most important benefits of the optimized use of resources.

The task of the scheduler in cloud computing is the scheduling of tasks and allocation of resources. Upon sending a task, the node selection process begins based on the information provided by the functionality of the nodes. Static algorithms do not take into account dynamic changes in node attributes at runtime. The static scheduler makes non-preemptive decisions and all task assignments are done before running the program. These decisions are based on information related to the tasks' execution time and the status of resources [8, 9]. Heterogeneous resources in hosts and the variety of requests in the cloud computing environment complicates the achievement of the precision required for predictions in this environment.

The use of optimization algorithms in this environment can increase the efficiency and productivity of cloud computing. Providing a scheduling optimizing method considering load balancing, response time, execution time, task migration, and so forth is of great importance, due to the complexity and variety of scheduling algorithms.

Some popular cloud computing infrastructures, such as Eucalyptus, use well-known greedy or round-robin scheduling algorithms. Simplicity is the main advantage of a greedy scheduling algorithm and the major drawback is low resource utilization. Xen uses a weighted distribution scheduling method and is least concerned about load balancing [10]. Optimization algorithms such as the dragonfly algorithm can be used for optimization application, such as network traffic. Polepally et al. in [11] eveloped a practical application for the dragonfly algorithm in the cloud computing domain. They used the dragonfly algorithm and a gravitational search algorithm to maintain load balancing in cloud computing. In the cloud environment, computational resources need to be scheduled in such a way as to maximize the use of resources from suppliers and users of the applications and provide for their needs at the lowest cost possible. Maximizing resource utilization in cloud suppliers, resource scheduling aims to minimize the user's cost for the applications.

In fact, the large scale of applications, the heterogeneity, and dynamism of resource characteristics of virtual machines, and the existence of various requests in the cloud computing environment make it difficult to achieve the precision required in the predictions of this environment. However, by using time estimation techniques and optimization algorithms, it is possible to increase the efficiency of cloud processing networks. The work scheduler in the cloud computing environment is used to ensure that user requests are properly scheduled for existing resources.

The current study was undertaken to examine load balances, service quality, and migration costs. Their features and abilities make them suitable to use the dragonfly and cat swarm optimization algorithms in cloud computing to optimize resource allocation and task scheduling. They were used herein to improve resource allocation in cloud computing while maintaining a load balance between resources.

Section 2 presents related works and section 3 describes the proposed method along with the offered architecture. In section 4 the obtained results are described and the conclusions presented in section 5.

2. Related works

Kansal et al. proposed an active clustering algorithm based on the grouping of similar nodes to provide a solution to load balancing in cloud computing. The most important advantages of this method are high-speed execution and processing tasks [12]. Sethi et al. presented a fuzzy logic based on turn-based load balancing in virtual machine environments in cloud computing in order to achieve better response time and better processing time. One of the advantages of this approach is that the load balancing algorithm is performed before the processing servers arrive [13]. James et al. proposed a method called turn-around. This algorithm assigns queries to virtual machines in turn [14]. Singh et al. proposed a method based on the Min-Min algorithm for scheduling tasks with load balancing [15]. Deinsh et al. proposed a load scheduling and load balancing method in cloud computing data centers based on the load balancing algorithm inspired by the colony honey bee algorithm. The main goal of this algorithm is to respond to requests and manage the best of virtual machines [16]. Kliazovich et al. proposed an optimal honey bee algorithm based method for balancing the load in cloud computing and scheduling tasks [17]. Mondal et al. put forward a parallel approach to the scheduling of data center tasks in cloud computing. One of the important issues they addressed in this study was the fast execution of the duties and cost of cloud owners [18]. Choudhary et al. presented a method called GSA based hybrid algorithm for bi-objective workflow scheduling in cloud computing. In their paper, they proposed a meta-heuristic based algorithm for workflow scheduling that considers minimization of makespan and cost. The proposed algorithm was a hybridization of the popular meta-heuristic, Gravitational Search Algorithm (GSA) and equally popular heuristic, Heterogeneous Earliest Finish Time (HEFT) to schedule workflow applications. They introduced a new factor called cost time equivalence to make the bi-objective optimization more realistic. They consider the monetary cost ratio (MCR) and schedule length ratio (SLR) as the performance metrics to compare the performance of their algorithm with other existing algorithms [19]. Elsherbiny et al. presented an extended Intelligent Water Drops algorithm for workflow scheduling in the cloud computing environment. They proposed a novel algorithm extending the natural-based Intelligent Water Drops (IWD) algorithm that optimizes the scheduling of workflows on the cloud. Their algorithm was implemented and embedded within the workflows simulation toolkit to achieve enhancements in the performance and cost [20]. Abazari et al. presented a method called MOWS: Multi-objective workflow scheduling in cloud computing based on a heuristic algorithm to respect security and performance together [21]. Zhou et al. presented a method called Minimizing cost and makespan for workflow scheduling in the cloud using fuzzy dominance sort based HEFT [22]. Razzaghzadeh et al. suggested a dynamic load balancing technique which relies on distributed queues [23]. This method uses colorful ants to distinguish the abilities of the human resources (HRs). Mapping was accomplished between the HRs and tasks by allocating a label to all HRs. Here, load balancing relies on poison distribution and mapping relies on exponential distribution.
Considering the speed of the algorithm and its efficiency in task scheduling, we used the Dragonfly optimization (DA) [24] along with the cat swarm optimization (CSO) [25] algorithms to improve the task scheduling and maintain load balancing in cloud computing data centers and presented CSO-DA.

3. The proposed method

In this section, we describe the steps of the proposed method and present a hybrid CSO-DA.

3.1. Select the best host with DA

It is important to maintain the load balancing between virtual machines in hosts in cloud computing. Load balancing in hosts means providing the same operational load between virtual machines in a host that ultimately ends in the load balancing of the host. The proposed algorithm, according to its capabilities, establishes the load balancing between virtual machines and selects a host with a lower overhead based on the amount of data overhead available among the total virtual machines running in the hosts of the data center. The process of selecting the host based on data overhead is shown in the following equations. In this process, it is important to calculate the expected response time, latency, packet sending time and allocated bandwidth, as shown in equations (3.1) to (3.6).

$$Responce\_Time{=}Fin{-}artt\_TDelay \tag{3.1}$$

The user login time is *artt_TDelay* and *Fin* time is the finishing time of the user request and the delay of the transfer, which is estimated using equation (3.2):

$$TDelay = Tlatency + Transfer \tag{3.2}$$

TDelay is transmission delay and *Tlatency* is the network latency and the *Transfer* is when the data of a request is transferred from one virtual machine to another destination and is estimated using equations (3.3) and (3.4).

$$Transfer = \frac{D}{Bwperuser} \tag{3.3}$$

$$Bwperuser = \frac{BWTotal}{Nr} \tag{3.4}$$

BWTotal is the total bandwidth available and *Nr* is the number of current user requests. Tracking the number of user requests between two virtual machines is maintained by *Nr*.
Based on these parameters, the fitting function of the dragonfly algorithm calculates the data overhead of each virtual machine shown in equation (3.5).

$$fitness_{VM_j} = ResponceTime\ _{VM_j} + TDelay\ _{VM_j} + Bandwidth\ _{VM_j} \tag{3.5}$$

Equation (3.6) calculates the data overhead.

$$H_{Li} = \sum_{j=1}^{n} fitness_{VM_j} \tag{3.6}$$

Therefore, the data overhead per host is calculated by the parameters based on equations (3.1) to (3.6) and the desired host is selected based on the finalized overhead.
The Algorithm 1 shows how to obtain the optimal host.

As seen in algorithm 1, this algorithm receives the number of hosts in the data center as the login argument. In lines 2 and 3, a variable is used to hold the hostname index number and data overhead per host. In line 4, the list of virtual machines is set to be completed in the process of executing the load balancing algorithm. In line 5, scrolling on all hosts so that the host can calculate the data overhead in each host and use it in subsequent processing. Line 6 calculates the number of virtual machines available per host. In line 8, all virtual machines are scanned and the amount of data overhead in each virtual machine is calculated according to (3.1) to (3.6). In line 9, the total data overhead of each virtual machine is computed and eventually, the data overhead of each host is also calculated. In lines 12 to 15, all the calculated overheads are compared and hosts with less data overhead are selected as target hosts. So with this simple strategy, while maintaining the load balancing between hosts, the optimal host to process the request, regarding energy consumption and resource allocation can be selected.

Algorithm 1 Pseudo-code of optimal host selection by proposed load balancing algorithm

```
1: procedure DA-FITNESS(Hosts)
2:     $BestHost \leftarrow 0$
3:     $HostIndex \leftarrow 0$
4:     $HostList \leftarrow \{Null\}$
5:     for <i = 1 To Hosts> do
6:         $VM_S \leftarrow GetVMS(Hosts)$
7:         $L_{VM} \leftarrow 0$
8:         for <j = 1 To $VM_S$> do
9:             $fitness_{VM} \leftarrow L_{VM} + (ResponceTime + TDelay + Bandwidth)_{VMj}$
10:        end for
11:       $HostList[i] \leftarrow LVM$
12:        if <HostList[i]is bettet than BestHost> then
13:            $BestHost \leftarrow HostList[i]$
14:            $HostIndex \leftarrow i$
15:        end if
16:    end for
17: end procedure
```

3.2. *Select best VM with CSO*

Cat swarm optimization (CSO) is an optimization heuristic algorithm based on the social behavior of cats [26]. It is inspiring by the social behavior of cats. Cats exhibit two modes of behavior [25]: 1) Seeking mode, in which cats do not move. They just stay in a certain position and sense for the next best move, thus having only state and not velocity. 2) Tracing mode, in which cats move to their next best positions with some velocity, representing how the cats chase their target. In performance comparison of swarm optimization algorithms, PSO with weighting factor shows better performance over simple PSO, but the experiments and results obtained from them show that CSO has better performance and results than PSO with weighting factor [26]. The Algorithm 2 shows the general steps for the CSO algorithm.

Algorithm 2 general steps of CSO algorithm

```
Step 1. Create j copies of the k$^{th}$ cat as represented by SMP.
Step 2. Modify the CDC dimension of each copy randomly.
Step 3. Evaluate the fitness of every copy.
Step 4. Find the best solutions among all copies that is the mapping with the minimum cost.
Step 5. Randomly choose a solution among them and replace it for the i$^{th}$ cat
```

Cats eliminate unnecessary use of energy, leading to efficient convergence towards a solution, by following these steps.

Proposed CSO algorithm for scheduling

In this paper we propose a customized algorithm based on CSO concept that aims to efficiently allocate resources to tasks in a cloud environment, focusing on minimizing the total cost incurred in the execution of all tasks. The proposed algorithm uses an initial population of N cats among which some are in seeking mode while others are in tracing mode, according to MR. Each cat represents a task-resource mapping, which is updated as per the mode that the cat is in.

Assessing the fitness value of the cats is to reduce the minimum cost of cat mapping. In each iteration, a new set of cats is chosen to be in tracing mode. The final solution, represented by the best position among the cats, gives the best mapping that has the minimum cost among all mappings.

1) Seeking mode:

This represents the majority of cats that search the global space while being in a resting state by intelligent position updating. Here the algorithm uses two basic factors - SMP and CDC. SMP (seeking memory pool) represents the number of copies to be made for each cat. CDC (count of dimension to change) defines how many of the allocations are to be altered in a single copy.

2) Tracing mode:

This represents the cats that are in a fast-moving mode and search the local space by moving towards the next best position with high energy. The general steps are as Algorithm 3:

Algorithm 3 The general steps of the tracing mode of implemented CSO algorithm

```
Step 1. Find velocity $v_{t+1}$ for the $i^{th}$ cat by (3.7)
Step 2. Update position of the cat as by (3.8)
Step 3. Check if the position goes out of the defined range. If so, assign the boundary value to the position.
Step 4. Assess the fitness value for the cats.
Step 5. Update the solution set with the best positions of the current iteration.
```

$$v_i^{t+1} = w{\times}v_i^t + rl \times cl \times (x_{best} - x_i^t) \tag{3.7}$$

where w is the inertia weight, $r1$ is a random number such that $0 \leq r1 \leq 1$ and $c1$ is the acceleration constant. $V_{i\ t}$ is the previous velocity, x_{best} is the best location and $x_i\ ^t$ is the current location.

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{3.8}$$

where x_i^t is the current position.

Therefore, using this algorithm, at any given moment, the most suitable virtual machines can be used to carry out the necessary processes for scheduling tasks in data centers, maintaining load balances, and providing a scheduling mechanism for tasks. Finding this virtual machine is shown in the flowchart of Figure 1.

4. Experimental results

The specifications of the computer we implemented the algorithm and results are as follows: 4 GB main memory Intel 8-core processor (Core^{TM}) i7 CPU Q 720@1.60 GHz. Table 1 shows the resources required by VMs and the tasks (processor, hard disk, RAM, etc.) used in this method.

The results obtained in this paper are compared with [16] and [27]. In order to compare the results, we have considered scenarios and examined the criteria. The criteria used are:

Execution time: The time between receiving and ending the execution of a request by a VM. The measurement unit of the execution time is the millisecond. The cost of execution time is calculated as the equation. (4.1):

$$EMT = \sum_{i=1}^{N} \frac{TaskExecutionTime_i}{NumberOfTasks} \tag{4.1}$$

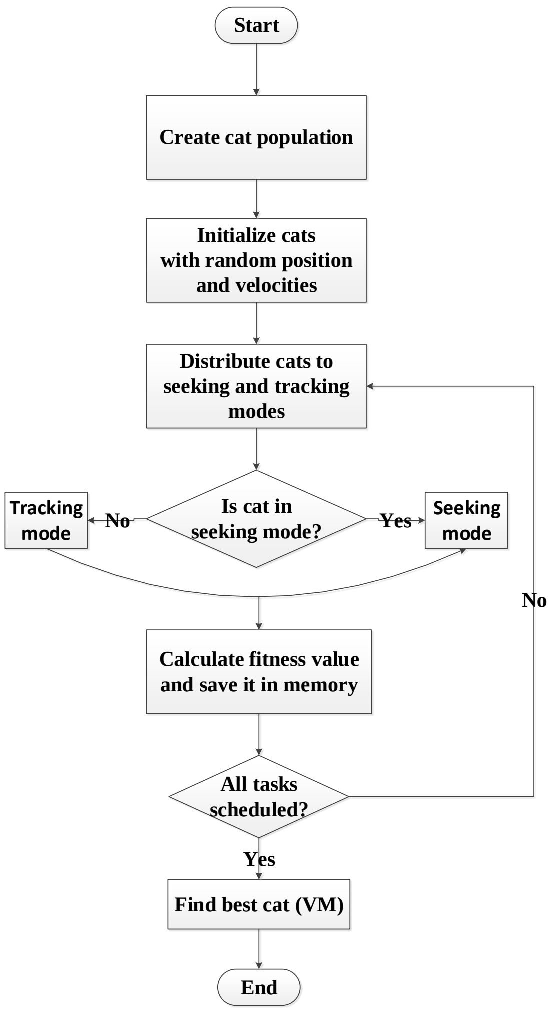
Start
Create cat population
Initialize cats with random position and velocities
Distribute cats to seeking and tracking modes
Is cat in seeking mode?
No
Yes
Tracking mode
Seeking mode
No
Calculate fitness value and save it in memory
All tasks scheduled?
Yes
Find best cat (VM)
End

Figure 1: Flowchart of general steps for the implemented algorithm

Table 1: Resources required by VMs and tasks

Factor	Levels
Disk size	8GB and 16GB
Network type	Megabit and Gigabit
Memory RAM capacity	512MB and 1024MB
VMs number	5, 20, and 40
vCPUs number	2
Tasks Number	10. 20 and 500

Response time: The time it takes from the moment a request is received by the VM until the processing of that request is completed. The measurement unit of response time is the millisecond. The cost of response time is calculated as the equation (4.2):

$$Response\ time = NumberOfTasks \times Time \tag{4.2}$$

Migrating tasks: This process occurs when the current virtual machine is not suitable to run a task, and the task has to migrate to another virtual machine.

With regard to the above scenarios, the scenarios for implementing the CSO-DA algorithm are defined as:

1. 10 tokens and 5, 20 and 40 virtual machines
2. 20 tokens and 5, 20 and 40 virtual machines
3. 100 tokens and 5, 20 and 40 virtual machines
4. 200 tokens and 5, 20 and 40 virtual machines
5. 500 tokens and 5, 20 and 40 virtual machines

Based on the above scenario, the results are examined and compared.

4.1. Evaluation of execution time

One of the most important parameters that are highly considered in scheduling tasks and resource allocation problems in a cloud computing system is the task execution time. This criterion is calculated based on CPU time and represents the efficiency of the scheduling algorithm. Table 2 shows the results of the execution time for scheduling and allocating of resources for 10 to 500 tasks and 5, 20 and 40 VMs. The aim of maintaining the load balance in cloud computing using the dragonfly optimization algorithm was compared with methods presented in [16] and [27]. As shown in Table 2, the execution time achieved using the CSO-AD algorithm in the proposed method is less than for the other methods and tasks were performed in less time. The execution time comparison is shown in Figure 2.

Table 2: Comparison of execution time of proposed algorithm with other methods.

	Number of tasks					
Number of VMs=5	10	20	40	100	200	500
Dragonfly	0.27	0.50	0.31	1.36	1.59	1.40
ACO	0.45	0.75	0.81	1.54	1.84	1.90
PSO	0.54	0.87	0.98	1.63	1.96	2.07
	Number of tasks					
Number of VMs=20	10	20	40	100	200	500
Dragonfly	0.27	0.50	0.31	0.92	1.15	0.96
ACO	0.45	0.75	0.81	1.10	1.40	1.46
PSO	0.54	0.87	0.98	1.19	1.52	1.63
	Number of tasks					
Number of VMs=40	10	20	40	100	200	500
Dragonfly	0.27	0.50	0.31	0.71	0.94	0.75
ACO	0.45	0.75	0.81	0.89	1.19	1.25
PSO	0.54	0.87	0.98	0.98	1.31	1.42

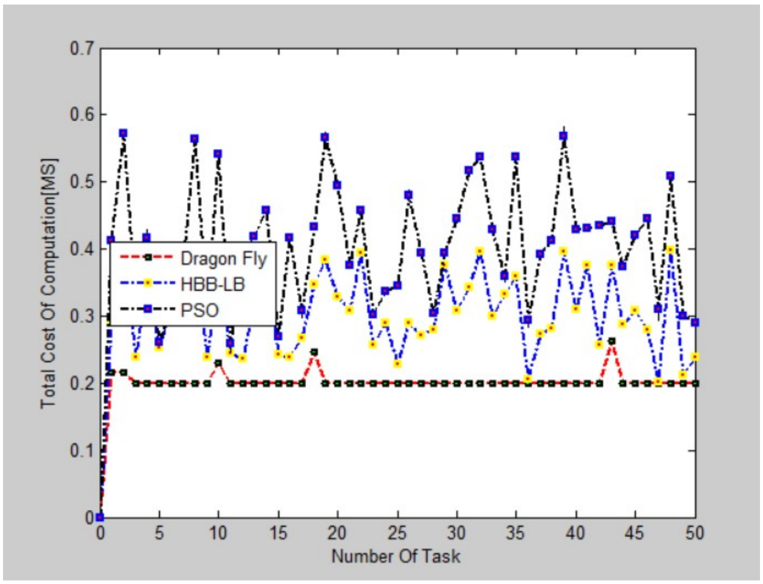

Figure 2: Comparison of execution time of proposed algorithm with other methods [24].

4.2. Evaluation of response time of requests

Response time is another criterion for evaluating the efficiency of allocation of cloud resources. Obviously, a lower the response time to a request for a resource allocation algorithm means better performance by the algorithm and better load balancing by the VMs. Table 3 shows the response time obtained using the CSO-DA optimization algorithm to schedule tasks between 10 to 500 tasks for 5, 20 and 40 VMs compared with [16] and [27]. As shown in Table 3, the total response time of the VMs to the tasks decreased as the number of VMs increased, but this downward trend does not continue endlessly. With a widespread increase in the number of VMs, only the cost of the cloud service provider increases.

Table 3: Comparison of response time of proposed algorithm with other methods.

	Number of tasks					
Number of VMs=5	10	20	40	100	200	500
Dragonfly	0.44	0.54	0.81	1.79	1.89	2.16
ACO	1.02	1.75	2.81	2.37	3.10	4.16
PSO	1.24	2.87	4.98	2.59	4.22	6.33
	Number of tasks					
Number of VMs=20	10	20	40	100	200	500
Dragonfly	0.44	0.54	0.81	1.16	1.26	1.53
ACO	1.02	1.75	2.81	1.74	2.47	3.53
PSO	1.24	2.87	4.98	1.96	3.59	5.70
	Number of tasks					
Number of VMs=40	10	20	40	100	200	500
Dragonfly	0.44	0.54	0.81	1.00	1.10	1.37
ACO	1.02	1.75	2.81	1.58	2.31	3.37
PSO	1.24	2.87	4.98	1.8	3.43	5.54

Table 4: Comparison of number of migrations of tasks by proposed algorithm with other methods

	Number of tasks					
Number of VMs=5	10	20	40	100	200	500
Dragonfly	1	1	2	4	5	9
ACO	2	2	3	7	10	19
PSO	3	3	4	10	13	28
	Number of tasks					
Number of VMs=20	10	20	40	100	200	500
Dragonfly	0	1	1	2	4	6
ACO	1	2	2	5	7	15
PSO	2	3	3	8	10	17
	Number of tasks					
Number of VMs=40	10	20	40	100	200	500
Dragonfly	0	0	1	1	2	5
ACO	1	2	2	4	5	12
PSO	2	2	3	7	8	15

As the number of migrating tasks in the cloud decreases, the better is the load balancing between the VMs. As the execution time and the response time of the tasks increase, the number of task migrations increase. Table 4 compares the number of tasks migrating between VMs in the CSO-DA algorithm and the methods presented by [16] and [27] for 10 to 500 tasks and 5, 20, and 40 VMs. The results show that considering the number of repetitions, the rate of migrating tasks from one VM to another in the proposed method is less than for the other methods presented in [16] and [27]. As the number of VMs increased, the number of migrations of tasks decreased, but the cost of the system also increased sharply with an increase in the number of VMs.

Figure 3 shows the decrease in the migration of tasks in the VMs and the consequent improvement in load balancing for the proposed method. As shown in Figure 3, the number of migrating tasks performed after several repetitions reached zero while maintaining the load balance. In the first repetitions, a relatively large number of tasks migrated, but gradually the number of migrations fell to zero. The proposed algorithm performed better than the other methods as the number of virtual machines and data centers increased, although the efficiency of the proposed method decreases with an increase in the number of VMs and data centers. As can be seen in Figure 3, the number of migrating tasks reaches the zero value after the load balancing.

5. Conclusion

In this paper, the efficiency of the CSO-DA optimization algorithm was examined based on the following criteria: runtime, response time, task migration and load balance. The results established that the hybrid CSO-DA algorithm is significantly more efficient, compared to other methods when it comes to task scheduling, load balancing, and resource allocation. Table 5 shows the results we obtained.
So, by these results of the above tables, it can be proved that CSO-DA algorithm provides significant improvements in task scheduling, load balancing, and resource allocations when comparing to other method.

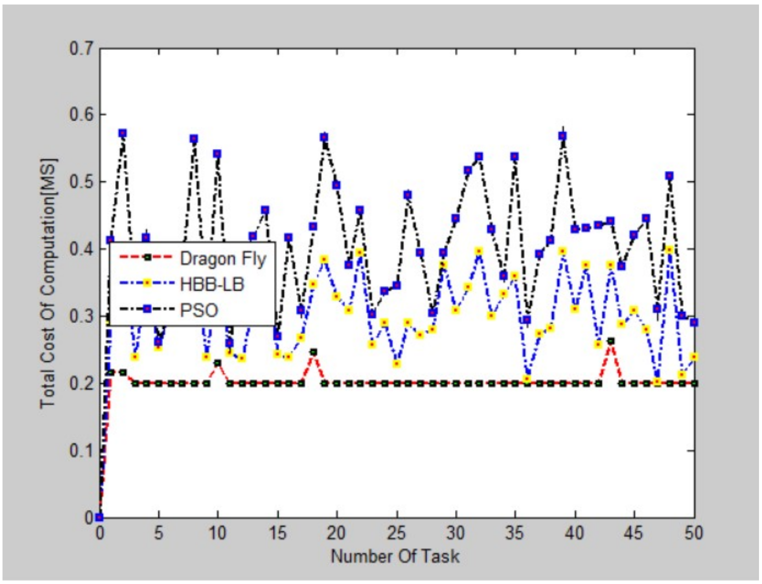

Figure 3: Reducing migration of tasks with proposed algorithm by maintaining load balance between VMs.

Table 5: Final comparison of dragonfly algorithm and other methods.

metrics	**CSO-DA**	**ACO**	**PSO**	**Improvement compared to ACO**	**Improvement compared to PSO**
Run-time average	0.723	1.033	1.16	41%	47%
Response time average	1.035	2.298	3.468	31%	39%
Task migration average	2	6	8	25%	42%

References

[1] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future generation computer systems*, vol. 28, no. 5, pp. 755–768, 2012.

[2] S. K. Panda and P. K. Jana, "Uncertainty-based qos min–min algorithm for heterogeneous multi-cloud environment," *Arabian Journal for Science and Engineering*, vol. 41, no. 8, pp. 3003–3025, 2016.

[3] S. B. Shaw and A. Singh, "A survey on scheduling and load balancing techniques in cloud computing environment," in *Computer and Communication Technology (ICCCT), 2014 International Conference on*, pp. 87–95, IEEE, 2014.

[4] P. Mell, T. Grance, *et al.*, "The nist definition of cloud computing," 2011.

[5] A. Pourghaffari, M. Barari, and S. Sedighian Kashi, "An efficient method for allocating resources in a cloud computing environment with a load balancing approach," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 17, p. e5285, 2019.

[6] H. Zhu and H. Wang, "New deadline-aware energy-consumption optimization model and genetic algorithm under cloud computing," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 30, no. 03, p. 1659006, 2016.

[7] A. K. Sidhu and S. Kinger, "Analysis of load balancing techniques in cloud computing," *International Journal of computers & technology*, vol. 4, no. 2, pp. 737–741, 2013.

[8] X. Evers, I. Herschberg, D. Epema, and J. de Jongh, "A literature study on scheduling in distributed systems," *Delft University of Technology*, 1992.

[9] A. A. Rajguru and S. Apte, "A comparative performance analysis of load balancing algorithms in distributed system using qualitative parameters," *International Journal of Recent Technology and Engineering*, vol. 1, no. 3, pp. 175–179, 2012.

[10] S. C. MOHARANA, S. SAMAL, A. R. SWAIN, and G. B. MUND, "Dynamic cpu scheduling for load balancing

in virtualized environments," *Turkish Journal of Electrical Engineering & Computer Sciences*, 2018 (Accepted paper).

[11] V. Polepally and K. S. Chatrapati, "Dragonfly optimization and constraint measure-based load balancing in cloud computing," *Cluster Computing*, pp. 1–13, 2017.

[12] N. J. Kansal and I. Chana, "Cloud load balancing techniques: A step towards green computing," *IJCSI International Journal of Computer Science Issues*, vol. 9, no. 1, pp. 238–246, 2012.

[13] S. Sethi, A. Sahu, and S. K. Jena, "Efficient load balancing in cloud computing using fuzzy logic," *IOSR Journal of Engineering*, vol. 2, no. 7, pp. 65–71, 2012.

[14] J. James and B. Verma, "Efficient vm load balancing algorithm for a cloud computing environment," *International Journal on Computer Science and Engineering*, vol. 4, no. 9, p. 1658, 2012.

[15] S. Singh *et al.*, "Load balancing algorithms in cloud computing environment.," *International Journal of Advanced Research in Computer Science*, vol. 9, no. 2, 2018.

[16] P. V. Krishna, "Honey bee behavior inspired load balancing of tasks in cloud computing environments," *Applied Soft Computing*, vol. 13, no. 5, pp. 2292–2303, 2013.

[17] D. Kliazovich, S. T. Arzo, F. Granelli, P. Bouvry, and S. U. Khan, "e-stab: Energy-efficient scheduling for cloud computing applications with traffic load balancing," in *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*, pp. 7–13, IEEE, 2013.

[18] B. Mondal, K. Dasgupta, and P. Dutta, "Load balancing in cloud computing using stochastic hill climbing-a soft computing approach," *Procedia Technology*, vol. 4, pp. 783–789, 2012.

[19] A. Choudhary, I. Gupta, V. Singh, and P. K. Jana, "A gsa based hybrid algorithm for bi-objective workflow scheduling in cloud computing," *Future Generation Computer Systems*, vol. 83, pp. 14–26, 2018.

[20] S. Elsherbiny, E. Eldaydamony, M. Alrahmawy, and A. E. Reyad, "An extended intelligent water drops algorithm for workflow scheduling in cloud computing environment," *Egyptian informatics journal*, vol. 19, no. 1, pp. 33–55, 2018.

[21] F. Abazari, M. Analoui, H. Takabi, and S. Fu, "Mows: multi-objective workflow scheduling in cloud computing based on heuristic algorithm," *Simulation Modelling Practice and Theory*, vol. 93, pp. 119–132, 2019.

[22] X. Zhou, G. Zhang, J. Sun, J. Zhou, T. Wei, and S. Hu, "Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based heft," *Future Generation Computer Systems*, vol. 93, pp. 278–289, 2019.

[23] S. Razzaghzadeh, A. H. Navin, A. M. Rahmani, and M. Hosseinzadeh, "Probabilistic modeling to achieve load balancing in expert clouds," *Ad Hoc Networks*, vol. 59, pp. 12–23, 2017.

[24] S. Mirjalili, "Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems," *Neural Computing and Applications*, vol. 27, no. 4, pp. 1053–1073, 2016.

[25] S. K. Saha, S. P. Ghoshal, R. Kar, and D. Mandal, "Cat swarm optimization algorithm for optimal linear phase fir filter design," *ISA transactions*, vol. 52, no. 6, pp. 781–794, 2013.

[26] S.-C. Chu, P.-W. Tsai, and J.-S. Pan, "Cat swarm optimization," in *Pacific Rim international conference on artificial intelligence*, pp. 854–858, Springer, 2006.

[27] M. Goyal and M. Aggarwal, "Optimize workflow scheduling using hybrid ant colony optimization (aco) & particle swarm optimization (pso) algorithm in cloud environment," *Int. J. Adv. Res. Ideas Innov. Technol*, vol. 3, no. 2, 2017.