Research Article

Kaushik Mishra and Santosh Kumar Majhi*

A binary Bird Swarm Optimization based load balancing algorithm for cloud computing environment

https://doi.org/10.1515/comp-2020-0215
Received Mar 20, 2020; accepted Nov 28, 2020

Abstract: Task scheduling and load balancing are a concern for service providers in the cloud computing environment. The problem of scheduling tasks and balancing loads in a cloud is categorized under an NP-hard problem. Thus, it needs an efficient load scheduling algorithm that not only allocates the tasks onto appropriate VMs but also maintains the trade-off amidst VMs. It should keep an equilibrium among VMs in a way that reduces the makespan while maximizing the utilization of resources and throughput. In response to it, the authors propose a load balancing algorithm inspired by the mimicking behavior of a flock of birds, which is called the Bird Swarm Optimization Load Balancing (BSO-LB) algorithm that considers tasks as birds and VMs as destination food patches. In the considered cloud simulation environment, tasks are assumed to be independent and non-preemptive. To evaluate the efficacy of the proposed algorithm under real workloads, the authors consider a dataset (GoCJ) logged by Goggle in 2018 for the execution of cloudlets. The proposed algorithm aims to enhance the overall system performance by reducing response time and keeping the whole system balanced. The authors have integrated the binary variant of the BSO algorithm with the load balancing method. The proposed technique is analyzed and compared with other existing load balancing algorithms such as MAX-MIN, RASA, Improved PSO, and other scheduling algorithms as FCFS, SJF, and RR. The experimental results show that the proposed method outperforms when being compared with the different algorithms mentioned above. It is noteworthy that the proposed approach illustrates an improvement in resource utilization and reduces the makespan of tasks.

Keywords: Binary BSO, Bird Swarm Optimization (BSO), Cloud Computing, DOI, Load balancing, Makespan, Resource utilization, Task scheduling

1 Introduction

Cloud computing is a new, popular technological trend in terms of computing, utilization of infrastructures, storage, networking, inclusive access services, and applications at various domains. The cloud delivers an internet-based scalable, elastic, flexible, utility, and metered computing platform based on a significant financial pay-per-use model. The use of cloud services is often driven by a service-level agreement (SLA) established between the cloud vendor and service consumers through negotiation. The physical resources can be encapsulated as abstract entities that deliver different levels of services to customers outside the cloud [1]. Cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers that provide those services [2]. The applications delivered are often referred to as Software-as-a-Service (SaaS), and the hardware and system software delivered are referred to as Infrastructure-as-a-service (IaaS) and Platform-as-a-Service (PaaS), respectively [3]. It is considered a cloud when these services are made available to different domains of people. When cloud services are delivered to people in a pay-as-you-go pricing model, it is called the public cloud. Private clouds are made available exclusively for an organization, and community clouds address the specific needs of a community. In contrast, hybrid clouds extract the benefits from both the public and private clouds.

On the flip side, cloud computing is becoming popular as the number of service consumers increases each day. The need for storage, capacity, and network are also rising with the increasing loads in data centers. Therefore, there is a need for technology to assimilate these rising consumer demands in the cloud. It could be possible through virtualization technology that enables the creation of an abstract

Kaushik Mishra: Department of Computer Science and Engineering, Veer Surendra Sai University of Technology, Burla, Odisha, India, 768018
***Corresponding Author: Santosh Kumar Majhi:** Veer Surendra Sai University of Technology Sambalpur, Odisha India; Email: smajhi_cse@vssut.ac.in

environment of the physical machine and has an efficient scheduling and load balancing techniques for distributing the loads among virtual machines (VMs).

During the scheduling process, tasks could be allocated onto the virtual machines (VMs) beyond their capacity, leading to an imbalance among the VMs. Hence, the system needs to check to determine whether the system is balanced. In some cases, resources are heavily loaded with tasks, but others are lightly loaded. Thus, there is a need to maintain an equilibrium among resources. This is referred to as load balancing in the cloud computing environment. Scheduling tasks and balancing the loads should be done in a way that ensures the whole system is balanced by uniformly dispersing the loads across VMs because it compels them to efficiently use the resources by diminishing the makespan and amplify the efficiency of the overall system.

It is believed that a robust load balancing will improve the Quality of Service (QoS) metrics such as throughput, makespan, reliability, response time, scalability, delay, and resource utilization [4]. Moreover, the problem of task scheduling and load balancing in the cloud are categorized under the NP-hard problem [5–7] because numerous scheduling parameters are involved. Depending on system situation, load balancing algorithms can be either static or dynamic [8, 9]. Dynamic algorithms are the most suitable load balancing algorithm when the requests and VMs are diverse [10]. Furthermore, static algorithms (*i.e.*, heuristics) are more problem-specific and dependent on the current state of the system, whereas dynamic (*i.e.*, meta-heuristics) are independent problem methods [11]. Others [12] have suggested that meta-heuristic algorithms are used to avoid the local optima problem. This algorithm allocates and reallocates the workloads to the VMs when there is a load imbalance. Although cloud computing is dynamic in nature, these algorithms are more suitable because they have more effective scheduling and load balancing techniques.

This article presents a population-based load balancing algorithm called the binary bird swarm optimization-based load balancing (BSO-LB) algorithm. This research primarily focuses on two problems. First, it runs a load balancing algorithm to trade-off the imbalance among loads across VMs. Once the system has found imbalance, the loads are migrated from heavily-loaded VMs to lightly-loaded VMs to create a balance among machines. Task migration is performed using the cosine similarity and compatibility of VMs with heavily-loaded tasks. The cosine similarity and the compatibility between the tasks of heavily-loaded VMs and lightly-loaded VMs are estimated. The task is migrated onto respective VMs if the compatibility is higher between them. After reaching uniformity in loads on the system level, task scheduling is initiated. The binary BSO algorithm maps tasks onto suitable VMs by identifying the possible best positions. Finally, the generated continuous solutions are transformed into discrete values using a small position value (SPV) rule. In a nutshell, the contributions of this paper are as follows:

- provides a fair distribution of loads among VMs using a nature-inspired meta-heuristic binary BSO (Bird Swarm Optimization) algorithm
- surveys related works about load balancing algorithms in cloud computing environments
- analyzes task mapping onto underloaded VMs, the compatibility between the tasks of overloaded VMs, and the available resources of underloaded VMs
- considers the binary position of each particle, a small position value (SPV) rule has been applied with each iteration
- provides an effective load balancing technique by considering a fitness function from the objectives of the cloud providers and cloud consumers simultaneously
- measures the efficacy by evaluating the proposed algorithm against other existing algorithms

The remainder of this paper is organized by section. Section 2 states the related research in the context of load balancing and task scheduling. Section 3 briefly introduces the BSO algorithm by highlighting the proposed load balancing formulation, followed by a step-by-step insight into the proposed algorithm. Section 4 demonstrates the simulation results and evaluates the proposed method, followed by a discussion on the findings. Finally, future research directions and concluding remarks are summed up in Section 5.

2 Related Work

Dynamic algorithms such as metaheuristic algorithms are efficient in tackling the dynamic nature of task scheduling in the cloud environment. In this section, related works in the context of load balancing in cloud computing are presented.

A proposed QoS-based min-min heuristic algorithm considers both high and low QoS requests and is processed separately [13]. Makespan has been taken as the QoS parameter, and this algorithm performs well compared to the basic min-min algorithm. The major drawback of the min-min algorithm is that it is unable to balance the loads among machines evenly. Others have proposed three algorithms for task scheduling in a heterogeneous multi-cloud envi-

ronment, including MCC, MEMAX, and CMMN [14]. The proposed MCC algorithm is a single-phase, and the other two algorithms have two-phase scheduling. They have taken makespan and average utilization as the performance metrics and validated their algorithms on benchmark functions and synthetic datasets as well [14]. Another study shows that their algorithm performs better than RR, CLS, CMMS, and CMAXMS. To achieve better utilization and efficiency, an optimized task scheduling algorithm that adapts to the advantages of other existing algorithms was proposed [15]. They compared the algorithm with some other well-known load balancing algorithms and found that it outperforms other algorithms.

Another study, inspired by the honey bee, presented a load balancing algorithm based on PSO using a single-objective function [16]. They evaluated the algorithm by choosing the VM with the highest compatibility for processing a request. They increased the utilization of resources by 22% while reducing the makespan by 33%. The current study is inspired by this approach. A honey bee-inspired load balancing technique (HBB-LB) is proposed to balance the loads amidst VMs [17]. They have taken the priorities of tasks into account for the selection of VM. But the low priority tasks remain in the queue for the longest time in order to execute. They used the Round Robin algorithm for assigning the tasks and migration mechanism to transfer tasks from heavily-loaded VM to underutilized VM. They have compared their HBB-LB algorithm with the FIFO, WRR, and DLB algorithms. An Ant Colony optimization-based load-balancing algorithm has been proposed that considers homogeneous tasks [18]. Their approach performed well in reducing makespan and response time but offers low scalability. Another study proposed a natural selection load balancing strategy using the Genetic Algorithm (GA) [19]. This algorithm tries to create possible solutions by mapping tasks onto VMs and evolving that solution over and over to find the solution that best positions onto VMs. The authors try to eliminate the inappropriate distribution of loads onto VMs in terms of its execution time.

Their algorithm offers low makespan and high resource utilization, but it is not reliable and is fault-tolerant. It is also insufficient for a heterogeneous environment. Aiming to reduce the total capital and power consumption of the server while migration, another study proposed a dynamic well-organized load balancing (DWOLB) algorithm using the GA [20]. This algorithm is meant for migrating the VMs at a server level. They claim that their proposed algorithm reduces the total cost and power consumption by approximately 25% compared to other existing load balancing techniques. Another multi-objective algorithm based on GA was proposed to improve the load balance and reduce the total power consumption in a cloud datacenter [21]. They effectively improved resource utilization while diminishing the power consumption of the datacenter. However, they have not considered the makespan, which is one of the prime factors while balancing the loads among VMs within a datacenter. By considering a heterogenous environment, authors in [22] have proposed a JAYA-based load balancing approach using the cloud analyst. Their algorithm considerably reduced the service time and response time but lacks in tackling the dynamically independent tasks. The algorithm is not validated for makespan and resource utilization which are the primary objectives of a load balancing procedure.

To get rid of sinking into local optima and have a better performance, a hybrid load balancing algorithm that integrates PSO with simulated annealing (SA) was proposed [23]. This algorithm aims to improve the utilization ratio of resources and convergence speed to the local optima. The algorithm shows better efficiency than SA, GA, and ACO. The current study compares the proposed algorithm with this algorithm. To improve the scheduling mechanism of VMs in cloud-based computing, authors in [24] have implemented a hybrid approach in combination of an ABC algorithm with the heuristic approach. The effectiveness of the algorithm is validated in both homogenous and heterogeneous environments by considering the makespan and load-balancing factors. This algorithm significantly reduced the above considered conflicting factors. This algorithm could have been validated against a real-world dataset by considering other QoS parameters like resource utilization, response time, etc. Finally, authors in [25] have developed a multi-criteria scheduling algorithm in an amalgamation of principles of quantum computing with the nature-inspired gravitational search algorithm for multiprocessor computing systems. In this research, both homogeneous and heterogeneous environments are considered to validate the efficacy of the proposed algorithm. It has been observed from the results that it outperforms for the considered scheduling objectives such as makespan, resource utilization, load-balancing. This algorithm could have improved in reducing energy consumption and for a workflow application.

3 BSO-based Load-balancing Algorithm

First, the authors explain the standard BSOA (Bird Swarm Optimization Algorithm) based optimization method, elu-

cidate the proposed load balancing algorithm, and then provide a detailed insight into the proposed algorithm.

3.1 The standard BSO

BSO is one of the recent inventions in the field of computational and swarm intelligence to solve the global optimization problem. This new population-based meta-heuristic algorithm was introduced by Meng *et al.* (2015) [26]. It is a bio-inspired optimization algorithm that strives to achieve global optima while getting over the local optima problem. It is inspired by the social behaviors among a flock of birds and their social interactions searching for food patches. It is based on three key mimicking behaviors of birds: foraging, vigilance, and flight. It is similar to other meta-heuristics algorithms that use a guided randomization mechanism to generate solutions with high diversity property [27]. These behaviors of birds are conceptualized by five rules [26–28].

According to rule one, each bird can either act as a forager or keep vigilance at one time. This process of foraging and keeping vigilance is modelled as a stochastic process. In the problem environment, tasks are removed from the overloaded VMs and are placed onto the underloaded VMs based on the availability of resources. The idea of foraging is related to the problem of mapping the tasks (as birds) onto the required VMs (as a destination food source). The behavior of keeping vigilance is analogous to the particle of having the best position. If a uniform random number in (0, 1) is smaller than P, $P \in (0, 1)$, a constant value, the bird will forage for food. Otherwise, the bird would continue vigilance.

Rule two states that during the foraging process, each bird searches for a food position based on their previous experience and the previous experiences of the flock of birds. The tasks keep updating their position according to their experience with respect to food position (destination VM). Figuratively, in the problem environment, particles keep updating their position with respect to their p_{best} and maps onto respective VM. This is formulated from Equation 1, and the initial position is calculated from Equation 2.

$$X_i^{k+1} = X_i^k + c \times rand_1 \times \left(p_{best_i} - X_i^k\right) + s \times rand_2 \times \left(g_{best} - X_i^k\right) \quad (1)$$

$$X_0^k = X_{min} + (X_{max} - X_{min}) \times rand \quad (2)$$

Note. X_0^k is the initial position of particle 0 at iteration k, X_{min} is the minimum value (–0.4), X_{max} is the maximum value (4.0), X_i^k is the current position of particle i at iteration k, X_i^{k+1} is the position vector of a particle i at iteration k, p_{best_i} is the best position of particle i, g_{best} is the global best position of the particle in the swarm, $rand_i$ are the random numbers between 0 and 1, i = 1,2 ($rand = rand_1 = rand_2 = 0.5$), C and S are the cognitive and acceleration coefficient, which are the two positive constant numbers (in our paper, $C = S = 1.5$).

According to rule three, while keeping vigilance, each bird would inevitably compete with each other to move closer to the centre of the flock rather than directly going to the centre. The particle has a lower fitness value and is considered the best position to move toward the centre of the swarm. This behavior of keeping vigilance could be correlated to the problem environment as the particles (tasks) with the least positions compete to move toward the forager with the best position. This is formulated from Equations 3, 4, and 5, respectively.

$$X_i^{k+1} = X_i^k + A_1 \left(p_{mean_j} - X_i^k\right) \times rand_a + A_2 \left(p_{best_l} - X_i^k\right) \times rand_b \quad (3)$$

$$A_1 = a_1 \times exp\left(-\frac{p_{best_i}}{p_{best_j}} \times N\right) \quad (4)$$

$$A_2 = a_2 \times exp\left(\left(\frac{p_{best_i} - p_{best_l}}{\left|p_{best_l} - p_{best_i}\right| + \epsilon}\right) \frac{N \times p_{best_l}}{p_{best_j} + \epsilon}\right) \quad (5)$$

Note. p_{mean_j} is the average position of the whole swarm, p_{best_l} is the best position of l^{th} particle ($l \neq i$, l is chosen from particle 1 to N), $rand_a$ is the random number between 0 and 1, $rand_b$ is the random number between –1 and 1, A_1 and A_2 are the indirect and direct effects induced by the surroundings, ai is the two positive constants (i = 1, 2{$i \in (0, 2)$}), p_{best_i} is the best fitness value of the i^{th} particle, p_{best_j} is the sum of the swarms' best fitness value, $\in$ is used to avoid zero-division error, and N is the total number of particles.

Rule four states that frequently, birds fly to another site in search of food. While flying, they often switch between producer and scrounger. The bird with the best fitness value would be the producer, and the bird with the worst fitness value would be the scrounger. The birds falling between the best and worst fitness values would randomly choose to be producer or scrounger. Producers and scroungers can be thought of as tasks (birds) having the best and worst fitness values. As the producers are always in search of a food source, it could be thought of as the tasks (producer) of overloaded VMs are always in search of underloaded VMs (food source). The scroungers are also the tasks (birds) with the least fitness values of underloaded VM who share the

same VM (food source) with the producer. The bird would switch their flight behavior at FQ interval of time. This flight behavior of birds is modelled in Equations 6 and 7.

$$X_i^{k+1} = X_i^k + rand_n(0, 1) \times X_i^k \tag{6}$$

$$X_i^{k+1} = X_i^k + \left(X_l^k - X_i^k\right) \times FL \times rand(0, 1) \tag{7}$$

Note. $rand_n(0, 1)$ is a random number drawn from Gaussian distribution random number with mean 0 and standard deviation 1, $X_l^k (l \neq i)$, l is chosen from particles 1 to N, and FL denotes the probability that the scrounger would follow the producer to search for food, where ($FL \in [0, 2]$).

Finally, rule five states that producers actively search for food sources, and the scroungers follow the producers. Upon arriving at the newly found food source, producers forage the food source again, and scroungers feed on the food source found by the producer. After being placed in one of the underloaded VMs (food source), the task (producer) still finds the best position and forages the food source again. The scroungers maintain the best position in one of the underloaded VMs (food source) found by the producer. This searching and feeding behavior of producer and scrounger can be separated in Equations 6 and 7.

The flowchart of the BSO based load-balancing algorithm is depicted in Figure 1.

3.2 The proposed load balancing Problem Formulations

An excellent approach to load balancing can reduce the request waiting time while maximizing the utilization of resources. It can prevent the VMs from either getting overloaded or underloaded and increases the uniformity in loads among VMs.

The load balancing approach in this study is inspired by the improved particle swarm optimization (PSO) based load balancing [16] and honey-bee inspired load balancing (HBB-LB) [17]. Distributing loads and searching for food by

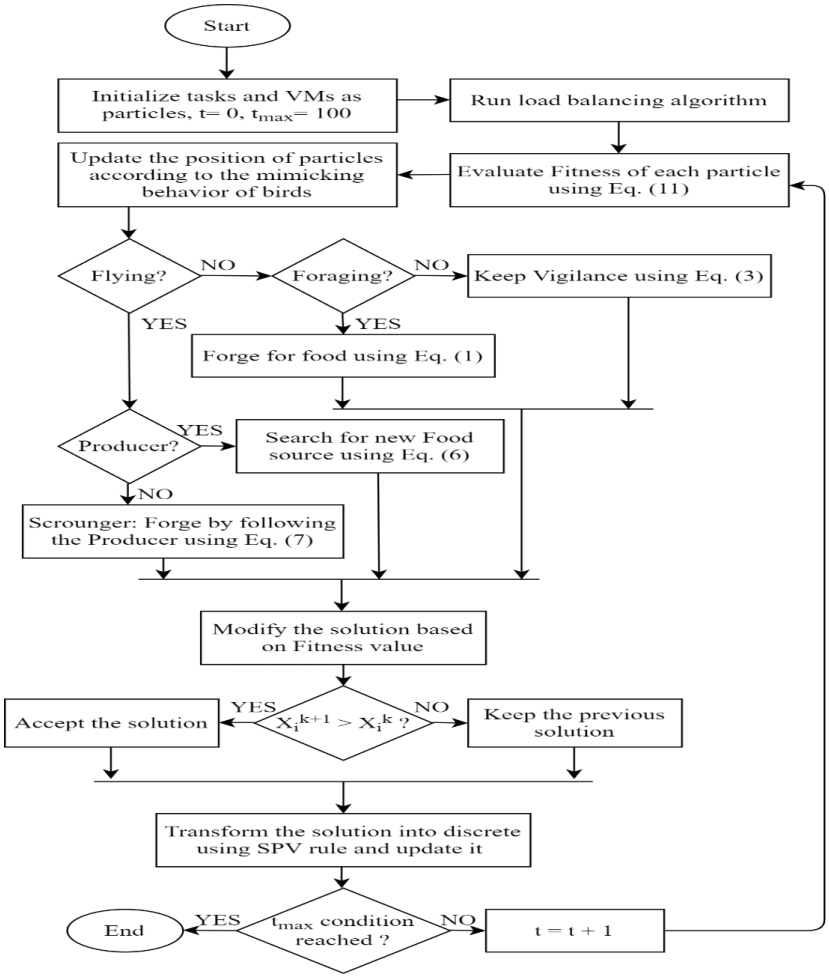


Figure 1: Flowchart of the BSO-based load balancing

swarm are correlated. Each bird in the swarm resonates as a particle in the cloud environment. Distributing the tasks among VMs is similar to how birds explore food sources. Empty food sources or already explored food sources act like an overloaded VM. So, there is a need to find another food source with the available resources and for finding a new underloaded VM for migrating the tasks. The fitness value of all particles is evaluated through the fitness function specified for a given problem to find the best position of a particle. The best position keeps updating at the end of every iteration. For instance, each particle in the cloud has its own fitness value; based on the best fitness value, tasks are assigned among VMs.

The load balancing problem is formulated using the following definitions.

Task Set: Consider a task set $T = \{T_1, T_2, T_3, \ldots, T_n\}$ of independent and non-preemptive tasks, where T_i, $1 \le i \le n$ is the i^{th} task with a million sets of instructions (MI) in l_i.

VM Set: Let a set of $VM = \{VM_1, VM_2, VM_3, \ldots, VM_m\}$ where VM_j, $1 \le i \le m$ is deployed under the number of hosts.

QoS Performance Metrics: QoS refers to throughput, response time, processing time, latency, availability, reliability, resource utilization, Degree of Imbalance (DOI), makespan, and power so that tasks can run in time and without any delay. These play a very vital role in measuring the efficiency of any algorithm. In this paper, we have considered makespan, response time, resource utilization, and degree of Imbalance (DOI) as the performance metrics.

Task Completion Time: the time taken to execute a task T_i by a VM_j that is computed by the difference between the start time and finish time [36]. It is denoted in Equation 8.

$$T_{CT_{ij}} = FT\left(T_j\right) - ST(T_j) \tag{8}$$

Makespan: refers to the maximum completion time of a task T_i among the VMs [29]. $T_{CT_{ij}}$ is the completion of a task T_i by VM_j, as depicted in Equation 9.

$$Makespan = \max\left\{T_{CT_{ij}} | i = 1, 2, \ldots, n; j = 1, 2, \ldots, m\right\} \tag{9}$$

Utilization: the degree of utilization of VMs [37, 38]. The objective of load balancing is to maximize the utilization of resources in order to minimize the makespan. These two terms are associated with a reverse linear relationship. The average utilization of all VMs is calculated using Equation 10 [30], where $m$is the total number of VMs.

$$Average\ Utilization_{VM} = \frac{\sum_{i=1}^{n} T_{CT_{ij}}}{makespan \times m} \tag{10}$$

Response Time: is the time taken to respond to the users' incoming request. The proposed method is taken as efficient if the response time is low and is measured in ms. Equation 11 calculates the response time where n is the total number of users' incoming request [31].

$$RT = n \times T_{CT_{ij}} \tag{11}$$

Fitness Function: the fitness function for our proposed load balancing algorithm to evaluate the fitness value of particles. It is a problem specific. Our objective is to maximize resource utilization while minimizing the makespan of the tasks and load imbalance among VMs. Therefore, the authors consider a single objective function by keeping the aforementioned objective into account. It is worth noting that smaller the fitness value, so the particle has a better position. Hence, the fitness function f_{val} is defined using Equation 12.

$$f_{val} = \frac{1}{Makespan} \times Average\ Utilization_{VM} \tag{12}$$

The load balancing problem is to map the tasks set T on the VM set V (f_{val}: $T \rightarrow V$) in a cloud such that the following objectives must satisfy: (1) the overall makespan should be minimized; (2) the average utilization of resources should be maximized by utilizing resources efficiently, and (3) loads of the system should be uniformly distributed among VMs to ensure a balanced system. These objectives are formulated using the aforementioned QoS parameter definitions.

3.2.1 Loads of a VM

The total length of tasks that are assigned to a VM is called load [17, 29, 32]. The load of every VM at time t is calculated by the number of tasks assigned to VM at time t divided by the service rate of VM at time t. The load of a VM is denoted using Equation 13.

$$L\left(VM_i, t\right) = \frac{NT(T, t)}{SR\left(VM_i, t\right)} \tag{13}$$

The load of all the VMs is calculated using Equation 14.

$$L = \sum_{i=1}^{m} L\left(VM_i, t\right) \tag{14}$$

The average load of the system is estimated using Equation 15, where m is the total number of VMs.

$$Average\ System\ Load = \frac{1}{m}\sum_{i=1}^{m} L\,(VM_i, t) \tag{15}$$

3.2.2 State of a VM group

To find the state of a VM group, the load of every VM will be compared against the average system load of the overall system. Based on the comparison, the state of the VM is identified as either underloaded ($L\,(VM_i, t) < Average\ System\ Load$), overloaded ($L\,(VM_i, t) > Average\ System\ Load$), or balanced ($L\,(VM_i, t) = Average\ System\ Load$).

3.2.3 Degree of Imbalance (DOI)

The degree of imbalance of VMs is the barometer to find the imbalances of tasks among VMs [18]. It is measured using Equations 16 and 17 [4, 30], where T_{max} and T_{min} are the maximum and minimum completion time of tasks T_i among all VMs. Further, T_{avg} is the average of all tasks T_i of VMs. L is the length of total instruction, PE_{num_i} is the number of processing elements in the i^{th} VM, and PE_{MIPS_i} are the million instructions per second of the i^{th} VM.

$$DOI = \frac{T_{max} - T_{min}}{T_{avg}} \tag{16}$$

$$T_i = \frac{L}{PE_{num_i} \times PE_{MIPS_i}} \tag{17}$$

The degree of imbalance in the system is gauged by identifying the lack of deviations in terms of loads at the system level using the standard deviation (σ). To find the state of the VM group, it is essential to find whether the system is balanced. If the value of the standard deviation is less than or equal to the threshold value (TS_h) that ranges between 0 and 1, then the system is balanced. Otherwise, the system is either overloaded or underloaded. The value of the threshold (TS_h) depends on the average system load due to the maximum capacity of the whole system. The load of the entire system cannot exceed this maximum capacity. Equation 18 is a barometer to estimate the deviation on the whole system load and triggers the desired load balancing operation.

$$\sigma = \sqrt{\frac{1}{m}\sum_{i=1}^{m} (PT_j - PT)^2} \tag{18}$$

Note. PT_j and PT is the processing time of all tasks on a VM and the processing time of the entire VM, respectively.

The processing time of all the tasks on a VM (PT_j) is expressed using Equation 19, and the processing time of all VMs is denoted using Equation 20.

$$PT_j = \frac{L(VM_i, t)}{Average\ System\ Load} \tag{19}$$

$$PT = \sum_{i=1}^{m} PT_j \tag{20}$$

3.2.4 Tasks Migration

The authors have adopted the following method, which is inspired by prior research [33] for the sake of migration of tasks onto underloaded VMs. CPU, storage, and BW are the resources usage pattern of each and every task T_i denoted as vector $\vec{T_i}$. Vector $\vec{T_{R_{used}}}$ is used to express the consumption of total resources by n tasks and vector $\vec{T_{R_{avail}}}$ is used to denote the available resources of underloaded VMs. Whereas $\vec{T_R}$ expresses the vector of total resources of underloaded VMs. These can be computed using Equations 21 and 22.

$$\vec{T_{R_{used}}} = \sum_{i=1}^{n} \vec{T_i} \tag{21}$$

$$\vec{T_{R_{avail}}} = \vec{T_R} - \vec{T_{R_{used}}} \tag{22}$$

To estimate the compatibility between tasks of overloaded VMs, VMs are represented as a vector $\vec{T_O}$, and the available resources of underloaded VMs are represented using the cosine similarity [16, 33] in the form of $angle$. The value of $angle$ indicates that greater similarity exists between tasks $\vec{T_O}$ and $\vec{T_R}$ if there a smaller value of $angle$. Compatibility [16] between task and the VM can be estimated with the help of the value of $angle$ and the resource utilization of that VM on which task is going to be mapped. Compatibility is represented as θ, and the value of α is set to 0.5 in this paper. The authors need to find the value of $angle$ at each iteration to formulate the compatibility between VM tasks and resources. The authors used this method in the load balancing approach to prevent the overloaded condition in VM and make an efficient migration of tasks to suitable underloaded VMs to trade-off the uniformity amidst VMs. It is based on the task-resource usage pattern vector and available resource usage pattern vector. $angle$ and $Compatibility(\theta)$ are calculated using Equations 23

and 24.

$$angle = \cos^{-1}\left(\frac{\vec{T_O} \times \vec{T_{R_{avail}}}}{\left|\vec{T_O}\right| \left|\vec{T_{R_{avail}}}\right|}\right) \tag{23}$$

$$\theta = \alpha \times angle + (1 - \alpha) \times \frac{T_{CT_{ij}}}{Makespan} \tag{24}$$

3.2.5 Binary BSO

In this contribution, the Small Position Value (SPV) rule [34, 35] is used to transform the generated continuous solutions into discrete solutions. Since the basic BSO is a continuous optimization technique, it will not be able to generate binary solutions for the problem like load scheduling, which is a binary in nature. Thus, it needs to be mapped to a binary version to solve the load assignment problem in the cloud environment.

The population of each particle of this problem is initialized with a position and fitness value. For a problem of n tasks, we represent each particle as an N-dimensional vector as $X_i^j = \left(X_1^1, X_2^2, X_3^3, \ldots, X_n^m\right)$ where $X_i (i \in \{1, 2, 3, \ldots, n\})$ is the number of tasks going to schedule on $X^j (j \in \{1, 2, 3, \ldots, m\})$ VMs. The position of the particle is represented by $X_i^k = \left(X_i^1, X_i^2, X_i^3, \ldots, X_i^n\right)$, where X_i^k represents the position of i^{th} particle with respect to the n dimensions. Applying this SPV rule, a particle's continuous position value $X_i^k = \left(X_i^1, X_i^2, X_i^3, \ldots, X_i^n\right)$ will be converted into discrete permutation sequences denoted as $S_i^k = \left(S_i^1, S_i^2, S_i^3, \ldots, S_i^n\right)$, where S_{ij}^k is the assignment of tasks implied by the particle i in the processing order at iteration k with respect to the j^{th} dimensions. To find the destination resources, the authors convert sequences S_i^k into resource vector $R_i^k = \left(R_i^1, R_i^2, R_i^3, \ldots, R_i^n\right)$, where R_i^k is represented as the n dimensional task processing on the R_i^k resources. It is computed using Equation 25.

$$R_i^k = S_i^k \; mod \; m, \tag{25}$$

where m is the total number of VM.

Permutation sequences S_i^k is estimated from the values of particles' position X_i^k. Smaller is the X_i^k value, so is the first sequence value is assigned to that particle. For instance, among the five particles, particle four has the smallest position value X_i^k. So, particle four will be assigned the value of one as the dimension ranges from 1 to n. Furthermore, the next smallest particle will be assigned the next smallest position value of two, and so on. Next, the resource position vector R_i^k is computed by taking the mod of S_i^k and m (total number of VM).

The values of X_i^k, S_i^k, and R_i^k are computed at each iteration, and the position of particles are updated simultaneously. This procedure of load balancing will keep going until it meets its stopping or termination criteria. In this paper, the authors have taken up to 100 iterations as the stopping or termination point (t_{max}).

3.2.6 Illustration of general task scheduling

The authors have illustrated a general task scheduling using the following examples. For illustration, a total of six tasks are processed on three VMs. To find the completion time of each task, the CloudSim is used [39, 40]. Table 1 shows the completion time of each task on three consecutive runs. On each iteration, tasks were assigned to respective VMs separately. The second column of Tables 2, 3, and 4 show the assigned tasks on the respective VMs. Based on the completion time, the maximum completion time is chosen as a Makespan, and then the resource utilization of each VM is calculated. Furthermore, the average resource utilization (ARU) is calculated by taking the average of resource utilization of the entire VM. The degree of imbalance (DOI) is estimated for the whole system using Equations 16 and 17.

In the first scheduling, the makespan is 8.31, and the ARU is 0.82. In the second and third scheduling, the makespans are 7.67 and 7.30, whereas the values of ARU are 0.83 and 0.92, respectively. Therefore, there is a gradual reduction in the makespan while maximizing the ARU. Hence, an efficient load balancing algorithm can reduce the makespan while increasing the utilization of resources. The value of DOI varies based on the algorithms used for scheduling the tasks and balancing the loads. A considerable amount of graphs can be depicted when the number of

Table 1: Completion time of tasks on each scheduling

	TASK1	TASK2	TASK3	TASK4	TASK5	TASK6
1st Run	3.75	4.00	2.29	3.24	2.98	4.31
2nd Run	3.60	3.12	4.18	3.49	2.62	2.28
3rd Run	3.10	4.00	4.22	2.80	4.20	3.00

Table 2: The first task scheduling

VMs	Assigned Tasks	Completion Time	Makespan	Resource Utilization (RU)	Average RU (ARU)	DOI
VM1	T3, T4	2.29+3.24=5.53	8.31	5.53 ÷ 8.31 = 0.66	(0.66 + 0.80 + 1.00) ÷ 3 = 0.82	0.57
VM2	T1, T5	3.75+2.98=6.73		6.73 ÷ 8.31 = 0.80		
VM3	T2, T6	4.00+4.31=8.31		8.31 ÷ 8.31 = 1.00		

Table 3: The second task scheduling

VMs	Assigned Tasks	Completion Time	Makespan	Resource Utilization (RU)	Average RU (ARU)	DOI
VM1	T2, T6	3.12 + 2.28 =5.40	7.67	5.40 ÷ 7.67 = 0.70	(0.70 + 0.81 + 1.00) ÷ 3 = 0.83	0.59
VM2	T1, T5	3.60 +2.62 =6.22		6.22 ÷ 7.67 = 0.81		
VM3	T3, T4	4.18 +3.49 = 7.67		7.67 ÷ 7.67 = 1.00		

Table 4: The third task scheduling

VMs	Assigned Tasks	Completion Time	Makespan	Resource Utilization (RU)	Average RU (ARU)	DOI
VM1	T1, T3	3.10 + 4.20 = 7.30	7.30	7.30 ÷ 7.30 = 1.00	(1.00 + 0.95 + 0.95) ÷ 3 = 0.96	0.40
VM2	T2, T6	4.00 + 3.00 = 7.00		7.00 ÷ 7.30 = 0.95		
VM3	T5, T4	4.20 + 2.80 = 7.00		7.30 = 0.95		

tasks increases. The above example shows how the reduction in makespan could lead to better resource utilization using an effective load balancing technique.

3.3 The proposed Binary BSO-LB algorithm

INPUT: $T = \{T_1, T_2, T_3, \ldots, T_n\}$,
$VM = \{VM_1, VM_2, VM_3, \ldots, VM_m\}$
OUTPUT: Best possible mapping of tasks onto VMs with a balanced system
BEGIN
1. Define particle and initialize particles' position using Equation 2
   for each particle,
   Create an N-dimensional vector
   $VM = \{VM_1, VM_2, VM_3, \ldots, VM_m\}$ where VM_j ($j \in \{1, 2, \ldots, m\}$) represents the number of VM on which task T_i ($i \in \{1, 2, \ldots, n\}$) is going to be processed,
   Calculate Completion time (TTC), makespan, Average resource utilization, and Response time using Equations 8, 9, 10 and 11,
   end for;
2. ***for*** each particle,
   Estimate the load using Equations 13, 14 and 15,
   end for;
3. Find state of the VM group based on Equation 15,
4. Task Migration based on VM group
   /* initiate migration mechanism from OverloadedVM to UnderloadedVM */
   while (UnderloadedVM! = ∅)
   Calculate the total resources used (TR_{used}) and total resources available (TR_{avail}) of all the UnderloadedVM using Equaions 21 and 22,
   Find the cosine similarity ($angle$) and compatibility (θ) of tasks of OverloadedVM with UnderloadedVM using Equations 23 and 24,
   end while
   while (OverloadedVM! = ∅ and UnderloadedVM! = ∅)
   Get CloudletList which need to transfer from OverloadedVM;
   for every Cloudlet belongs to CloudletList does Insert Cloudlet into UnusedCloudletList (UUCL);
   end for
   end while
   /* Sorting Cloudlet */
   while (UnderloadedVM! = ∅)
   Sort UnderloadedVM in ascending order based on respective loads,
   Rearrange the UnderloadedVM having the same state of loads in descending order based on Resource utilization;
   Sort OverloadedVM in descending order based on loads;
   end while
   /* Assigning Cloudlet to UnderloadedVM */
   while (OverloadedVM! = ∅ and UnderloadedVM! = ∅)
   for i = 0 to #(OverloadedVM)
   do

Assign $T_i(OverloadedVM) \rightarrow UnderloadedVM_j$ based on compatibility (θ) using Equation 24,
Check $L\left(VM_j, t\right) < Average\ System$;
end for
end while
5. ***for*** each particle,
do
Calculate fitness value using Equation 12,
if $\left(X_i^{k+1} > X_i^k\right)$
Set the current fitness value as the new best position,
end if
Calculate the position using Equations 1, 3, 4, 5, 6 and 7,
end for
6. /* Update Particles' position */
for each particle,
do
Update their position using Small Position Value (SPV) rule based on Equation 25;
end for
7. /* Estimate deviation on load */
Calculate the standard deviation on system load using Equation 18,
if ($\sigma \leq TS_h$)
The system is balanced;
else
Trigger load balancing;
8. Repeat steps (3) to (7) until reaching the equilibrium state of the system.

END

4 Performance Evaluation

In this section, the authors present the simulation setup and the experimental results conducted by various tests. A detailed analysis of the results is presented in this section.

4.1 Experimental Setup

The classes of CloudSim [39, 40] toolkit have been extended to simulate and model the cloud environment. This simulator creates a virtualized environment that supports on-demand provisioning. This simulator helps to model, simulate, and experiment with the cloud services and its applications [39]. For the experiment, the authors defined ten possible solutions for the algorithm and the maximum termination (t_{max}) criterion is set to 100 iterations. The algorithmic parameters for the proposed algorithm are presented in Table 5.

In the experimental setup, the authors created one datacenter that consists of four hosts, each capable of creating

Table 5: The algorithmic parameters

PARAMETER	VALUE
Number of the Candidate solution	10.00
Maximum iteration	100.00
C, S	1.50
rand, $rand_1$, $rand_2$	0.50
X_{max}	4.00
X_{min}	−0.40
$rand_a$	A random number between 0 and 1
$rand_b$	A random number between −1 and 1
a_1, a_2	A random number between 0 and 2
FL	A random number between 0 and 2
TS_h	0.85
α	0.50

Table 6: Host Technical Details

Host ID	Processing Cores	Speed, MIPS	RAM, GB	Storage, GB	BW, MIPS
1	1	3500	50	1024	102400
2	2	4000	100	1024	102400
3	3	4500	150	1024	102400
4	4	5500	200	1024	102400

Table 7: VM Technical Details

CPU	Number of Cores	Speed, MIPS	RAM, GB	Storage, GB	BW, MIPS
Core_i5_Extreme_Edition	1	1000	2	20	1024

instances of physical machines and sharing the resources among VMs. Each host includes some processing cores, RAM, storage, bandwidth (BW), and processing speed. The technical specifications of each host are shown in Table 6. Thirty-six VMs were considered that run on each host with distinctive characteristics. The VM technical details are shown in Table 7. The authors deemed real workloads to analyze the performance of this algorithm. The authors used a dataset that is GoCJ: Google cloud jobs dataset for distributed and cloud computing infrastructures by Google published in September 2018 [41]. This dataset contains 19 text files with a different number of jobs in a million instructions (MI) and stored in the Mendeley data repository. The authors considered one of the text files named GoCJ_Dataset_100.txt that comprised of 100 jobs, and each job is taken as a cloudlet regarding its length in MI. The number of cloudlets (20-100) with varying sizes of instructions, ranging from 1000 to 5000, is considered for the execution of respective VMs.

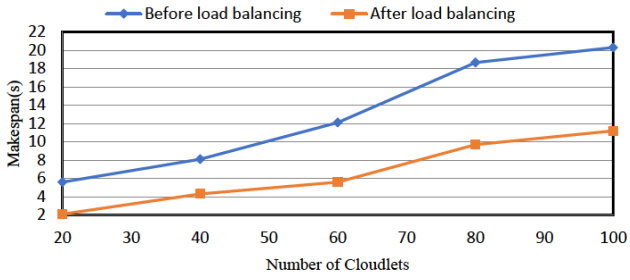
5 Result Analysis

5.1 Experimental Results

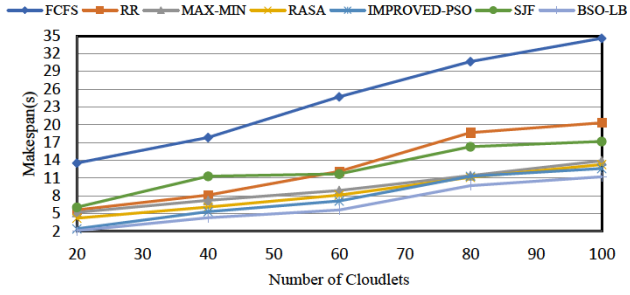
The proposed algorithm is simulated through a generalized framework by extending the classes of CloudSim [39, 40]. In this section, the authors present the analysis of the results based on the simulation done using CloudSim. To analyze the efficacy of the proposed algorithm, different algorithms such as (1) Round Robin (RR), (2) FCFS, (3) SJF, (4) MAX-MIN, (5) RASA and (6) Improved PSO have been considered for comparison. The results were obtained by noting down the mean values for each performance metrics by running each algorithm ten times.

In the following series of graphs, the experimental results are shown in terms of makespan, resource utilization, response time, and DOI. Figure 2(a) shows the comparison between makespan before and after load balancing using BSO-LB. The X-axis represents the number of cloudlets, and the Y-axis represents the makespan.

The proposed method is compared with other existing algorithms stated above in terms of makespan. According to Figure 2(b), the proposed method shows a better result for evenly balancing the load among nodes. Figure 3 represents the obtained values of the response time for a number of cloudlets for the different algorithms with the proposed algorithm. The results show that the proposed approach has a better response time compared to other algorithms. It is observed from the simulation results that the performance of other compared algorithms on response time gets increased with the increasing cloudlets. But the proposed method achieves a good performance over others. The comparison of resource utilization for existing algorithms with the proposed algorithm is depicted in Figure 4. The proposed BSO-LB algorithm performs better because it has the ability to efficiently utilize the resources by allocating the loads onto the respective VMs.



(a) Comparison of Makespan before and after load balancing



(b) Makespan Comparison of existing algorithms with BSO-LB

Figure 2

The authors present a reverse linear relationship between makespan and resource utilization in Figure 5. The graph shows that the proposed algorithm accomplishes maximum resource utilization while minimizing makespan. The proposed algorithm can significantly utilize the resources at maximum and considerably reduces the makespan.

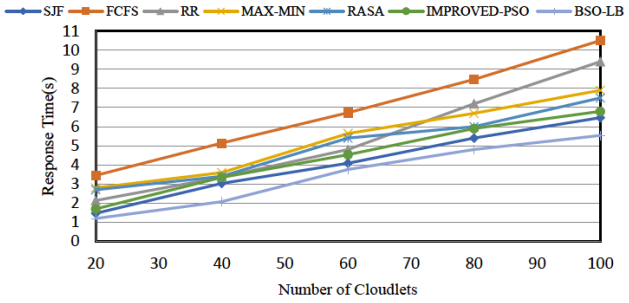


Figure 3: Comparison of algorithms in Response Time

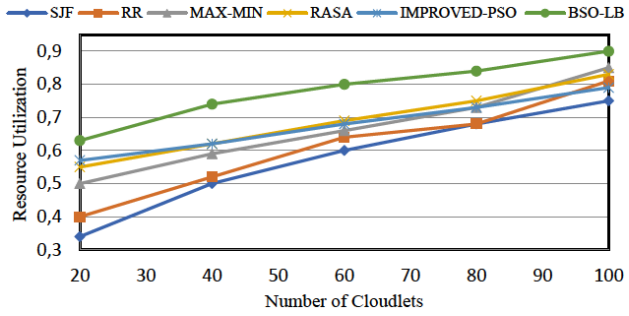


Figure 4: Resource Utilization Comparison

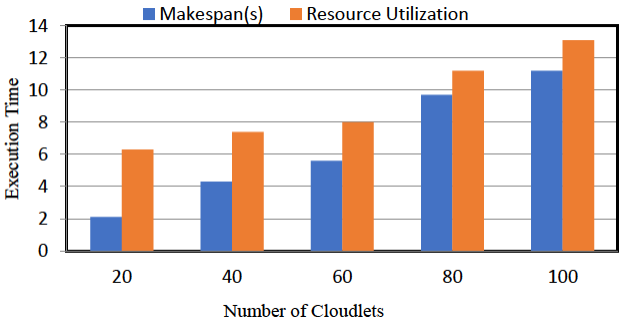
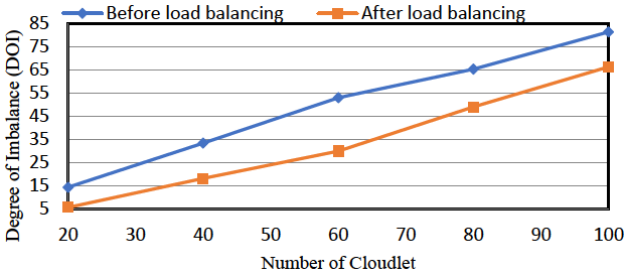
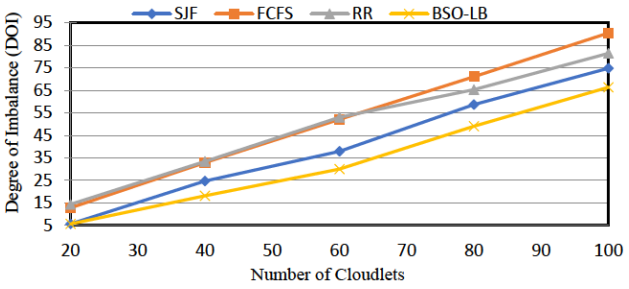


Figure 5: Comparison of Makespan and Resource utilization

The DOI between VMs before and after load balancing using BSO-LB is shown in Figure 6(a). The X-axis represents the number of cloudlets, and the Y-axis represents the degree of imbalance. It shows that the proposed algorithm balances the loads among VMs effectively, and the degree of imbalance is considerably reduced. Figure 6(b) illustrates the comparison between the DOI and other algorithms with the proposed BSO-LB algorithm. The DOI is less in the present algorithm compared to other algorithms.



(a) Degree of Imbalance (DOI) before and after load balancing using BSO-LB



(b) Degree of Imbalance (DOI) values obtained using three methods and proposed BSO-LB algorithm

Figure 6

6 Discussion

The authors propose an algorithm that combines the load balancing method with the meta-heuristic BSO algorithm. To analyze the performance of the proposed method, they compared it to existing algorithms (*e.g.*, FCFS, RR, SJF, MAX-MIN, RASA, Improved-PSO) for makespan, resource utilization, response time, and DOI. Cloudlets are assumed to be independent and non-preemptive for the simulation. The proposed method is used for the dynamic scheduling of independent tasks.

The Bird swarm Optimization (BSO) algorithm is used for mapping the cloudlets onto respective VMs. The mimicking behavior of birds is the perfect fit for the problem space. These behaviors divide the birds into two groups: producer and scrounger. The parameter *FL* must be chosen meticulously so that the scrounger can be distinguished from the producer. In the proposed BSO-LB algorithm, this is considered to be 0.5. The fitness value also plays a trivial role in the selection of producer and scrounger. The selection of fitness function and the proper distribution of tasks among VMs also significantly affect the overall performance of the system. Authors considered a single-objective fitness function in terms of makespan and average resource utilization. So, they considered a task-resource compatibility test to measure the compatibility of the tasks for the overloaded

VMs with the underutilized VMs based on their available resources. Thus, resource utilization was enhanced while also reducing the makespan, which is shown in Figure 5. Therefore, the proposed method is efficient in distributing the loads uniformly.

The authors also considered a small position value (SPV) rule to transform the continuously generated solutions to discrete solutions since the task scheduling problem is dynamic and requires binary values to represent the tasks-resources assignment in the problem space. Moreover, a proper, yet rugged use of the meta-heuristic technique combined with a load balancing technique, reduces the makespan. The process of migrating tasks from overloaded VMs onto underloaded VMs and using an effective load balancing technique improves the performance and efficiency of the system [42]. Figures 2(a) and 2(b) show that the makespan is considerably reduced by using the proposed algorithm when compared to other algorithms with varying cloud lengths. Figure 5 also depicts the relationship between makespan and resource utilization.

The response time is affected when the length of the cloudlets increases. The proposed algorithm uses a suitable allocation of cloudlets onto VMs through an effective load balancing technique. Hence, according to Figure 3, at some point, it may reach the level of SJF when the number of cloudlets is 60 and then eventually start decreasing. The study used various cloudlet lengths, ranging from 1000 to 5000, to analyze the response time and performance. Figure 3 also shows that the reduced makespan leads to better overall response time.

In the case of resource utilization, the efficiency of resources is greatly utilized due to the reduction of makespan and the fitness function used in the algorithm. The utilization of resources is better than other algorithms and is shown in Figure 4. The DOI also leads to an increase in makespan and decrease in resource utilization. Hence, it is essential to maintain the degree of balance between VMs. The results in Figures 6(a) and 6(b) depict the degree of imbalance before and after load balancing, and it is considerably reduced.

The simulation results demonstrate that the proposed method outperforms other algorithms while increasing the number of cloudlets. The simulated results show the reduced makespan, response time, and DOI while maximizing resource utilization and throughput.

7 Conclusion and Future Scope

Load balancing plays a crucial and yet significant role in the cloud computing environment. It often comes before an effective task scheduling mechanism to balance out the uneven distribution of loads in the system. An effective load balancing technique is necessary to utilize resources and reduce the makespan efficiently. With respect to the objective as mentioned above, the authors proposed a binary variant of the BSO-inspired load balancing technique based on the three mimicking behaviors of birds (*i.e.*, foraging, vigilance, and flight). These behaviors are analogous to the problem of mapping the tasks (as birds) onto the required VMs (as destination food sources) and the particles with the best position. Tasks are removed from the overloaded VMs and are placed onto the underloaded VMs based on the availability of resources. The flock of birds (tasks) keeps updating their position according to their experience with respect to food position (destination VM). The proposed binary BSO-LB algorithm initially evaluates the position of particles and keeps updating the position at each iteration. It is capable of minimizing the makespan and maximizing resource utilization by using a proper fitness function. To measure the compatibility between the tasks of overloaded VMs and resources of underloaded VMs, a task compatibility test has been carried out. In addition to it, a discrete position value has been formulated using an SPV rule for positioning the particles. The BSO-LB method has been compared and analyzed against other existing algorithms. It shows notable improvements over the compared algorithms for makespan, response time, and resource utilization. Moreover, the proposed algorithm is capable of handling independent and both preemptive and non-preemptive tasks in any cloud environment.

In the future, meta-heuristic based solutions could be considered for the load balancing problem with heterogeneous resources. This problem could be improved by applying an increasing number of tasks and VMs in heterogeneous environment. Other QoS performance metrics could also be considered to validate the effectiveness of the algorithm.

References

[1] Mishra S. K., Sahoo B., Parida P. P., Load balancing in cloud computing: a big picture, Journal of King Saud University-Computer and Information Sciences. 2020, 32(2), 149-58.

[2] Josep A. D., Katz R., Konwinski A., Gunho L. E. E., Patterson D., Rabkin A., A view of cloud computing, Communications of

the ACM, 2010, 53(4), 50-58. https://doi.org/10.1145/1721654.1721672

[3] Mell P., Grance T., The NIST definition of cloud computing, National Institute of Standards and Technology, 2011.

[4] Milan S. T., Rajabion L., Ranjbar H., Navimipoir N. J., Nature inspired meta-heuristic algorithms for solving the load-balancing problem in cloud environments, Computers & Operations Research, 2019, 110, 159-187. https://doi.org/10.1016/j.cor.2019.05.022

[5] Li W., Tordsson J., Elmroth E., Virtual machine placement for predictable and time-constrained peak loads, In: International Workshop on Grid Economics and Business Models. Springer, Berlin, Heidelberg, 2011, 120-134.

[6] Ibarra O. H., Kim C. E., Heuristic algorithms for scheduling independent tasks on nonidentical processors", Journal of the ACM (JACM), 1977, 24(2), 280-289.

[7] Ullman J. D., NP-complete scheduling problems, Journal of Computer and System sciences, 1975, 10(3), 384-393.

[8] Shah N., Farik M., Static load balancing algorithms in cloud computing: Challenges & solutions, International Journal of Scientific & Technology Research, 2015, 4(10), 365-367.

[9] Mishra K., Majhi S. K., A state-of-Art on cloud load balancing algorithms, International Journal of computing and digital systems, 2020, 9(2), 201-220. http://dx.doi.org/10.12785/ijcds/090206

[10] Chaharsooghi S. K., Kermani A. H. M., An effective ant colony optimization algorithm (ACO) for multi-objective resource allocation problem (MORAP), Applied mathematics and computation, 2008, 200(1), 167-177. https://doi.org/10.1016/j.amc.2007.09.070

[11] Bala A., Chana I., A survey of various workflow scheduling algorithms in cloud environment, In: 2nd National Conference on Information and Communication Technology (NCICT), 2011, 26-30.

[12] Kalra M., Singh S., A review of metaheuristic scheduling techniques in cloud computing, Egyptian informatics journal, 2015, 16(3), 275-295. https://doi.org/10.1016/j.eij.2015.07.001

[13] He X., Sun X., Von Laszewski G., QoS guided min-min heuristic for grid task scheduling, Journal of Computer Science and Technology, 2003, 18(4), 442-451.

[14] Panda S. K., Jana P. K., Efficient task scheduling algorithms for heterogeneous multi-cloud environment, The Journal of Supercomputing, 2015, 71(4), 1505-1533. https://doi.org/10.1007/s11227-014-1376-6

[15] Mittal S., Katal A., An optimized task scheduling algorithm in cloud computing, In: 6^{th} IEEE International Conference on Advanced Computing (IACC), 2016, 197-202.

[16] Ebadifard F., Babamir S. M., A PSO-based task scheduling algorithm improved using a load-balancing technique for the cloud computing environment, Concurrency and Computation: Practice and Experience, 2018, 30(12), 1-16. https://doi.org/10.1001/cpe.4368

[17] LD D. B., Krishna P. V., Honey bee behavior inspired load balancing of tasks in cloud computing environments, Applied Soft Computing, 2013, 13(5), 2292-2303. https://doi.org/10.1016/j.asoc.2013.01.025

[18] Li K., Xu G., Zhao G., Dong Y., Wang D., Cloud task scheduling based on load balancing ant colony optimization, In: IEEE Sixth Annual ChinaGrid Conference, 2011, 3-9. https://doi.org/%2010.1109/ChinaGrid.2011.17

[19] Dasgupta K., Mandal B., Dutta P., Mandal J. K., Dam S., A genetic algorithm (ga) based load balancing strategy for cloud computing, Procedia Technology, 2013, 10, 340-347. https://doi.org/10.1016/j.protcy.2013.12.369

[20] Vanitha M., Marikkannu P., Effective resource utilization in cloud environment through a dynamic well-organized load balancing algorithm for virtual machines, Computers & Electrical Engineering, 2017, 57, 199-208. https://doi.org/10.1016/j.compeleceng.2016.11.001

[21] Zhang M., Ren H., Xia C., A Dynamic Placement Policy of Virtual Machine Based on MOGA in Cloud Environment, In: IEEE International Symposium on Parallel and Distributed Processing with Applications and IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC), 2017, 885-891. https://doi.org/10.1109/ISPA/IUCC.2017.00135

[22] Mohanty S., Patra P. K., Ray M., Mohapatra S., An Approach for Load Balancing in Cloud Computing Using JAYA Algorithm, International Journal of Information Technology and Web Engineering (IJITWE), 2019, 14(1), 27-41.

[23] Zhan S., Huo H., Improved PSO-based task scheduling algorithm in cloud computing, Journal of Information & Computational Science, 2012, 9(13), 3821-3829.

[24] Kruekaew B, Kimpan W., Enhancing of Artificial Bee Colony Algorithm for Virtual Machine Scheduling and Load Balancing Problem in Cloud Computing, International Journal of Computational Intelligence Systems, 2020, 13(1), 496-510.

[25] Thakur AS, Biswas T, Kuila P., Binary quantuminspired gravitational search algorithmbased multicriteria scheduling for multiprocessor computing systems, JOURNAL OF SUPERCOMPUTING, 2020

[26] Meng X. B., Gao X. Z., Lu L., Liu Y., Zhang H., A new bio-inspired optimization algorithm: Bird Swarm Algorithm, Journal of Experimental & Theoretical Artificial Intelligence, 2016, 28(4), 673-687. https://doi.org/10.1080/0952813X.2015.1042530

[27] Aljarah I. *et al.*, Evolving neural networks using bird swarm algorithm for data classification and regression applications, Cluster Computing, 2019, 1-29. https://doi.org/10.1007/s10586-019-02913-5

[28] Lin M., Zhong Y., Lin J., Lin X., Discrete Bird Swarm Algorithm Based on Information Entropy Matrix for Traveling Salesman Problem, Mathematical Problems in Engineering, 2018, 1-15. https://doi.org/10.1155/2018/9461861

[29] Ebadifard F., Babamir S. M., Barani S., A dynamic task scheduling algorithm improved by load balancing in cloud computing, In: 6th International Conference on Web Research (ICWR), IEEE, 2020, 177-183

[30] Mapetu J. P., Chen Z., Kong L., Low-time complexity and low-cost binary particle swarm optimization algorithm for task scheduling and load balancing in cloud computing, Applied Intelligence, 2019, 49(9), 3308-3330.

[31] Priya V., Kumar C. S., Kannan R., Resource scheduling algorithm with load balancing for cloud service provisioning, Applied Soft Computing. 2019, 76, 416-424.

[32] Polepally V., Chatrapati K. S., Dragonfly optimization and constraint measure-based load balancing in cloud computing, Cluster Computing. 2019, 1-13.

[33] Nanduri R., Maheshwari N., Reddyraja A., Varma V., Job aware scheduling algorithm for mapreduce framework, In: IEEE Third International Conference on Cloud Computing Technology and Science, 2011, 724-729. https://doi.org/10.1109/CloudCom.2011.112

[34] Tasgetiren M. F., Sevkli M., Liang Y. C., Gencyilmaz G., Particle swarm optimization algorithm for single machine total weighted tardiness problem, In: Proceedings of the 2004 Congress on Evolutionary Computation, IEEE, 2004, 2, 1412-1419.

[35] Zhang L., Chen Y., Yang B., Task scheduling based on PSO algorithm in computational grid, In: Sixth International Conference on Intelligent Systems Design and Applications, IEEE, 2006, 2, 696-704.

[36] Chakravarthi K. K., Shyamala L., Vaidehi V., TOPSIS inspired cost-efficient concurrent workflow scheduling algorithm in cloud, Journal of King Saud University-Computer and Information Sciences, 2020. https://doi.org/10.1016/j.jksuci.2020.02.006

[37] Khorsand R., Ghobaei-Arani M., Ramezanpour M. A., Self-learning fuzzy approach for proactive resource provisioning in cloud environment, Software: Practice and Experience, 2019, 49(11), 1618-1642.

[38] Rafieyan E., Khorsand R., Ramezanpour M., An adaptive scheduling approach based on integrated best-worst and VIKOR for cloud computing, Computers & Industrial Engineering, 2020,140, 106272.

[39] Buyya R., Ranjan R., Calheiros R. N., Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities, In: international conference on high performance computing & simulation, IEEE, 2009, 1-11.

[40] Calheiros R. N., Ranjan R., Beloglazov A., De Rose C. A., Buyya R., CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Software: Practice and experience, 2011, 41(1), 23-50. https://doi.org/10.1002/spe.995

[41] Hussain A., Aleem M., GoCJ: Google cloud jobs dataset for distributed and cloud computing infrastructures, Data, 2018, 3(4), 38. https://doi.org/10.3390/data3040038

[42] Jena U. K., Das P. K., Kabat M. R., Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment, Journal of King Saud University-Computer and Information Sciences, 2020. https://doi.org/10.1016/j.jksuci.2020.01.012