Spider Monkey Optimization Algorithm for Load Balancing in Cloud Computing Environments

Sawsan Alshattnawi and Mohammad AL-Marie
Department of Computer Science, Yarmouk University, Jordan

Abstract: *Scheduling of tasks is one of the main concerns in the Cloud Computing environment. The whole system performance depends on the used scheduling algorithm. The scheduling objective is to distribute tasks between the Virtual Machines and balance the load to prevent any virtual machine from being overloaded while other is underloaded. The problem of scheduling is considered an NP-hard optimization problem. Therefore, many heuristics have been proposed to solve this problem up to now. In this paper, we propose a new Spider Monkeys algorithm for load balancing called Spider Monkey Optimization Inspired Load Balancing (SMO-LB) based on mimicking the foraging behavior of Spider Monkeys. It aims to balance the load among virtual machines to increase the performance by reducing makespan and response time. Experimental results show that our proposed method reduces tasks' average response time to 10.7 seconds compared to 24.6 and 30.8 seconds for Round Robin and Throttled methods respectively. Also, the makespan was reduced to 21.5 seconds compared to 35.5 and 53.0 seconds for Round Robin and Throttled methods respectively.*

Keywords: *Cloud computing, load balancing, metaheuristic optimization, spider monkeys optimization, tasks scheduling.*

Received April 1, 2020; accepted January 6, 2021
https://doi.org/10.34028/iajit/18/5/13

1. Introduction

Cloud computing consists of a pool of IT resources scattered over the entire world and available for paying users through a pay-as-use model [10, 24]. The increasing number of cloud customers has led to an increased load over these resources. The tasks must be allocated efficiently to ensure that there is no overloaded machine while other is idle.

Load balancing was defined by Patil *et al*. [19] as "load balancing allocates load (work) across multiple devices intending to use maximum resources with the highest efficiency and minimum response time along with preventing single resource overload". Therefore, Load Balancing (LB) has to allocate resources to achieve several critical measurements necessary to obtain high performance such as resource utilization, throughput, response time, and scalability.

LB is considered one of the NP-complete problems in the Cloud Computing environment [9, 22]. Many algorithms have been proposed to solve this problem. These algorithms can be divided into static, dynamic, and optimization-based algorithms [4, 23]. The static algorithms work for small variations of workload and don't take into account changes in load during run-time. The cloud computing environment is dynamic, and it needs dynamic algorithms for efficient scheduling and load balancing among Virtual Machines (VMs) [1, 21]. The nature-inspired algorithms are very efficient in solving dynamic real-time problems that could be hard to solve by classic methods. Therefore, Optimization methods in Artificial intelligence have been used to enhance and optimize load balancing concerning execution time.

Optimization-based algorithms are numerical methods using mathematical optimization. They adopted Metaheuristics that belong to the mathematical optimization family. They are stochastic methods based on a search model, attempting to find an optimal global solution among a set of feasible solutions. Swarm intelligence, mostly inspired by natural biological systems that simulate the cooperative behavior of an organized group of animals or insects, as they struggle to survive [19].

In this paper, we propose a load balancing approach inspired by the foraging behavior of spider monkeys [5]. Spider monkeys follow a dynamic and cooperative tactic for finding sufficient food sources for all members of the swarm (group). The foraging behavior of spider monkeys ensures that the process of searching for food should be as fast as possible and all members of the group should be fed. Thus, the Spider Monkey Optimization (SMO) algorithm is applied to optimize the load balancing problem in terms of distributing users' tasks among VMs to achieve the minimum response time and makespan time.

To the best of our knowledge, this is the first work that applies SMO in load balancing. Therefore, the key contribution of this work is:

- Proposing a mathematical model of the foraging behavior of Spider Monkey to be applicable for load balancing in cloud computing environments.

- Adopting a grouping strategy for both virtual machines and users' tasks to mimic the nature of food sources VMs and the nature of biological swarms.
- Testing and comparing the proposed method with other algorithms.

1.1. Spider Monkey Optimization

SMO algorithm [5] is a swarm intelligence-inspired algorithm based on the foraging behavior of spider monkeys. Spider monkeys are social animals in which live in groups and follow a particular living pattern in terms of communication and foraging for food. A group of spider monkeys always is led by a female leader, foraging for food in which should be enough for all group members. In case of insufficient food, the leader splits the group into slighter sub-groups which in turn foraging in different dimensions and regions to increase the opportunity of finding food sources. The leader of the whole group called the global leader while leaders of subgroups called local leaders.

The optimization process of the spider monkey algorithm is achieved throughout six phases:

1. Local leader phase: while a sub-group is foraging, members changing their positions based on information from the local leader and from group members.
2. Global leader phase: members of all sub-groups are changing their positions based on information from the Global leader and the local leader of the sub-group.
3. Global leader learning phase: the global leader hadn't changed his position by applying a greedy selection on all members to find the nearest spider monkey to a food source, and then his position will be the position of the global leader.
4. Local leader learning phase: same as the previous phase but within the domain of the sub-group.
5. Local leader decision phase: to avoid stagnation, if the local leader hadn't changed her position within a prespecified time, then all sub-group members will update their positions based on information from the global leader and the local leader.
6. Global leader decision phase: also, if the global leader didn't update her position within a prespecified time, then she divided the group into two sub-groups in the first iteration and for three sub-groups in the second iteration and so on until the maximum number of allowed groups is reached. If the swarm of spider monkeys has been split into the maximum groups' number and the position of the global leader hadn't changed, then she combines all sub-groups to form one group.

During foraging, spider monkeys follow a dynamic approach for finding the best food sources with the lowest possible time. Thus, SMO is an appropriate algorithm to be applied for solving the load balancing problem in cloud computing.

2. Literature Review

Many load balancing algorithms have been proposed. These algorithms can be classified according to the utilized approach as static, dynamic, and optimization-based methods.

2.1. Static Methods

In these methods, task distribution is made at compile-time; they don't consider the current load of the resources. Those methods do not need to monitor or acquire information about the behavior of the system [17]. One of the first simplest static algorithms is Round Robin (RR) [18]. It depends on dividing time into slices. It does not take into consideration the changing demands for the resources and the node's state. Another algorithm is Throttled Load Balancing Algorithm; it depends on a table for allocation and deallocation of nodes [11]. It distributes the load depending on the index inside the table based on a requested service. The problem with this method is that it fails to achieve the resource utilization criteria.

2.2. Dynamic Methods

In these Methods, the distribution of load depends on the current information of the system, which is changing over time. The distribution process happens at runtime [17]. Shortest Job First (SJF) presented in [15], aims to improve the scheduling efficiency of jobs' distribution, minimize the response time, and increase virtual machines' availability. It depends on the allocation priority of services to nodes taking into consideration the status of the current node. It gave better results than RR.

2.3. Optimization-based Methods

In the field of computation, optimization techniques could be mathematically based as Linear Programming techniques or techniques based on Meta-heuristics such as Ant colony and Bee colony optimization. Swarm intelligence, mostly inspired by biological systems, simulates the cooperative behavior of an organized group of animals or insects, as they struggle to survive [19]. Researches in different computational fields have been simulated the behavior of bees, ants, fireflies, and other creatures for solving different optimization problems [2, 16, 20, 25].

A load-balancing method inspired by Honeybee behavior was proposed in [12]. It focused on reducing execution time and waiting time of tasks in the queue taking into consideration tasks' priorities. They found significant improvement in makespan when compared to Weighted Round Robin (WRR), First in First out (FIFO), and dynamic load balancing. Some

modification is made over the bee colony algorithm to enhance load balancing over VMs, to minimize makespan and to decrease VM migration [25]. The algorithm is modified by adding two concepts, honeybees (overloaded VMs) and food sources (under loaded VMs). Experiments showed better results than the original bee colony algorithm.

The firefly algorithm is also used to maximize resource utilization and for better load balancing in cloud servers [8]. Two algorithms for dynamic load balancing, Fuzzy Logic, and Particle Swarm Optimization algorithm are combined [3] to reduce the makespan. The proposed model was tested using CloudSim and the results were compared with the results of First Come First Serve (FCFS) and Particle Swarm Optimization (PSO) methods.

A hybrid algorithm that combined three algorithms: ant colony, particle swarm, and honeybee algorithm presented in [29]. The proposed algorithm gave superior results when compared to the results of the artificial bee colony, ant colony optimization, and PSO algorithms. It reduced the makespan and improved resource utilization. The Scheduling Algorithm is a popular meta-heuristic that is easy to implement as indicated in [31], the authors solve this problem by PSO because it is easy to trap into a locally optimal solution. The work presented in [30] is more efficient than other traditional algorithms in terms of energy-saving and load-balancing. They use a new coalitional game approach for optimizing the energy efficiency of VM consolidation in heterogeneous cloud datacenters.

Partitioning the cloud into many parts with several nodes and using the bee colony algorithm to optimize the waiting time was the idea proposed in [7]. Simulation results proved that the Completion Time (CT) of overall tasks was less than CT in Honey Bee Behavior Inspired Load Balancing (HBB-LB), WRR, and FCFS algorithms.

Sundararaj [28] combined the bee and ant colony approaches to assign tasks to process in the cloud computing for the mobile users. The proposed approach deals with two-way mobile cloud computing with offloading technique. This technique uses the Ant Colony Optimization ACO algorithm to minimize the overhead problems and the delay in the response time.

Mansouri *et al*. [14] proposed a hybrid task scheduling approach by combining the modified particle swarm optimization and fuzzy theory. This algorithm used important issues for example speed of CPU, and total execution time in the fuzzy system for the calculation of the fitness.

Sreenu and Malempati [26] proposed a fractional gray wolf optimization technique with modification on the position of each task. This technique uses multi-objective task scheduling. The algorithm addressed multiple objectives such as the execution of time, cost, consumption of energy, and resource utilization.

Su *et al*. [27] proposed cost-efficient task scheduling that executed a large number of programs on the cloud. The researchers applied this algorithm through two heuristic approaches. This method dynamically combined tasks and the virtual machine. Also, the service providers of a cloud environment may rent the cloud resources as payable services.

On mimicking the dynamicity in the foraging behavior of spider monkeys in terms of the dynamic combining and splitting of the group members until feeding all of them. This mimicry is achieved through the continuous grouping and splitting of tasks according to the balance state of VMs (overloaded, balanced, underloaded) until assigning every task in the group to a VM. We develop our SMO-inspired Load Balancing to solve dynamic load balancing problems in cloud computing, Spider Monkeys (SMs) maintaining a dynamic foraging behavior by avoiding stagnation. Therefore, mimicking this smart behavior in the cloud environment will lead to optimizing load balancing in terms of dynamicity.

3. Spider Monkey Optimization Inspired Load Balancing

Cloud computing environments are empowered by Virtualization technology. Virtualization enables a single processing machine (such as a Server) to be turned into a network of virtual machines. The job of a load balancer is mapping the flow of clients' tasks into those VMs and to keep VMs in a balanced state. This could be achieved by reducing response time and makespan. Response time is the interval of time taken for a user request to be responded to, while makespan is the interval of time that elapses from the start of tasks processing to the end [12], it is called also completion time.

Considering the problem of load balancing as the problem of food foraging for spider monkeys, where users' tasks are spider monkeys and they should forage to find adequate food sources VMs. The general concept of the adopted Spider Monkey Optimization Inspired Load Balancing (SMO-LB) approach is based on mimicking the dynamicity in the foraging behavior of spider monkeys in terms of the dynamic combining and splitting of the group members until feeding all of them. This mimicry is achieved through the continuous grouping and splitting of tasks according to the balance state of VMs (overloaded, balanced, underloaded) until assigning every task in the group to a VM.

We develop our Spider Monkey Optimization-inspired Load Balancing to solve the dynamic load balancing problem in cloud computing, SMs maintaining a dynamic foraging behavior by avoiding stagnation. Therefore, mimicking this smart behavior in the cloud environment will lead to optimizing load balancing in terms of dynamicity.

To apply SMO in load balancing, we reconfigure the cloud environment in terms of grouping assumptions. The goal of our grouping assumption is to mimic the nature of food sources VMs and the nature of biological swarms. Thus, we follow a grouping strategy for both VMs and Users' Tasks (UTs). On one hand, VMs are grouped based on their processing capacity which depends on the allocated resources Random Memory acces (RAM), processing units, bandwidth) concerning the length of estimated traffic flow. Each VMs group represents a region of food sources with a direction to each food source VM. On the other hand, the flow of tasks is grabbed through groups, where each group represents a spider monkeys' group. This grouping strategy paves for the load balancer to reach an optimal balance state. Figure 1 shows the general framework of the SMO-LB.

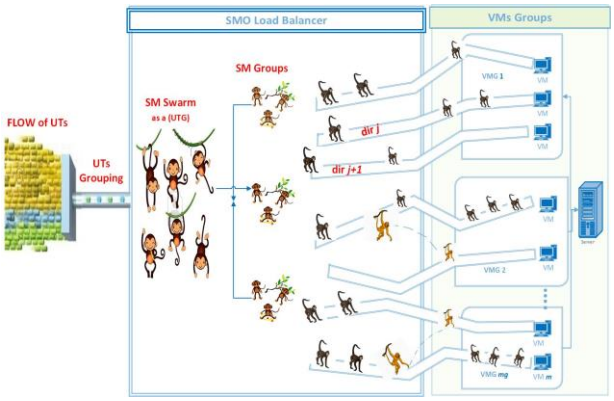

Figure 1. SMO-LB general framework.

3.1. Mathematical Model for Load Balancing

The mathematical model encompasses all necessary terms, formulas, and calculations for the SMO-LB algorithm.

Set of Virtual Machines: $VM = \{VM_1, VM_2, ..., VM_m\}$

Set of Users Tasks: $\boldsymbol{UT} = \{UT_1, UT_2, \ldots..., UT_t\}$

Set of VM groups: $VMG = \{VMG_1, VMG_2, \ldots..., VMG_{mg}\}$

Set of UT groups: $UTG = \{UTG_1, UTG_2, \ldots..., UTG_{tg}\}$

Group of VMs: $VMG_g = \{VM_i, \ldots..., VM_{ig} \mid VM_i \in VM\}$ $where: VMG_g \bigcap_{c=1}^{mg} VMG_c = \varnothing$

Group of UTs: $\boldsymbol{UTG_u} = \{UT_i, \ldots..., UT_{iu} \mid UT_i \in UT\}$ $where: UTG_u \bigcap_{c=1}^{tg} UTG_c = \varnothing$

Where, as stated earlier here, each VMG_g is corresponding to a region of multi food sources, while each UTG_u is corresponding to a group of foraging spider monkeys.

Processing Capacity (PC) of VM_j [1]:

$$PC_j = Pnum_j * Ppwr_j + Cspd_j \quad (1)$$

Where: $Pnum_j$: number of processing elements on VM_j,

$Ppwr_j$: millions instruction per second for all processors allocated for VM_j, and $Cspd_j$: communication bandwidth handling speed of VM_j

Processing Capacity of VMG**:**

$$PC_{VMG_g} = \sum_{j=1}^{ig} PC_j \quad (2)$$

Processing Capacity of the System:

$$PC_{Sys} = \sum_{j=1}^{m} PC_j \quad (3)$$

- Execution Time ET: ETij denotes the expected execution time of UT_i in VM_j, which can be defined as the load (length) of the task divided into machine speed PC [13]

$$ET_{ij} = \frac{UT_i\ Length}{PC_j} \quad (4)$$

UT_i Length: is the length of a user task i in million instruction

- Completion Time CT: CT_j denote the expected time for VM_j for executing all assigned UTs

$$CT_j = S_{.Curtime} + \sum_{i=1}^{n} ET_{ij} \quad (5)$$

Where $S._{curtime}$ is the current time of the simulator or the data center.

CT_{ij} denote the expected completion time of UT_i on VM_j

$$CT_{ij} = CT_j + ET_{ij} \quad (6)$$

Completion Time of VMs group: the expected execution time of all queued tasks to a particular VMs group, VMG_g

$$CT_{VMG_g} = \sum_{j=1}^{ig} CT_j \quad (7)$$

- Makespan: denote overall completion time of a UT [20]

$$\text{Makespan} = \max \begin{Bmatrix} CT_{ij} \mid i \in UT,\ i = 1, 2, \ldots, n\ and \\ j \in VM,\ j = 1, 2, \ldots, m \end{Bmatrix} \quad (8)$$

- Response Time: the amount of time taken between the submission of a service request and the first response.

Response Time of UT_i to be handled by VM_j:

$$RS_{ij} = CT_{ij} - (arr_t + delay_t) \quad (9)$$

Where: arr_t is the arrival time of the user request, $delay_t$ is the transmission delay time.

- State of Current Load (CL): $CL_{(VMj,t)}$ denotes workload of a VM_j at a given time *t*

$$CL_{(VM_j, t)} = \frac{Squeue.length_{VM\ j} * PT_j}{PC_j},\ at\ time\ t \quad (10)$$

Where Squeue.$length_{VMj:}$ is the number of UTs in VM_j service queue at time *t*.

The workload of VMs group at a given time *t*

$$CL_{(VMG_g,\,t)} = \sum_{j=1}^{ig} CL_{(VM_j,\,t)} \quad (11)$$

The workload of the whole system at a given time *t*

$$CL_t = \sum_{j=1}^{m} CL_{(VM_j,\,t)} \quad (12)$$

- Load Balance Measurement σ_t: denote the state of the system's workload CL compared with the system's PC at time *t*.

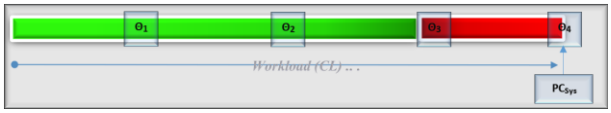

Figure 2. System workload.

To find this relationship, the systems performance range is divided into quarters. Each quartile represents a state of system performance. A Θ point is heading each quarter to make a load balancing decision, Θ1, Θ2, Θ3 indicating that the workload of the data center is 25%, 50%, 75% respectively as shown in Figure 2. For instance, if the total workload CL reached Θ3 then activate the load balancer. Multiple Θ points are due to achieving a more flexible load balancing decision making. However, when the workload exceeding Θ4 then the load balancing is not possible since it becomes broader than the system's processing capacity.

$$\sigma_t = PC_{Sys} - CL_{time\,t} \quad (13)$$

Algorithm 1: Load Balancing Decision

```
1: if σ_S.curtime ≥ Θ_4 then
2:     Load Balancing is not possible
3:     exit(1);
4: else:
5:     when σ_S.curtime = Θ_3 then:
           # Θ determined by system administrator
6:     activate Load Balancer
7:     Continue;
```

- dir_{ig}: direction of UTi toward VMGg → task UTi is at the service queue of VM_j, where $VM_j \in VMG_g$

Direction Boundaries:

$$Mini_g = Avg\left(\min\left\{RS_{ij} \mid VM_j \in VMG_g\right\},\ \min\left\{CT_{ij} \mid VM_j \in VMG_g\right\}\right) \quad (14)$$

$$Max_g = \frac{\sum_{j=1}^{ig}\left(PC_j - \mathrm{CL}_{(\mathrm{VM_j},\,\mathrm{t})}\right)}{UT_i\ Length} \quad (15)$$

The maximum boundary of dir_{ig} represents the available processing capacity of VMG_g to serving *x* of UTs before getting overloaded, in other words, for spider monkeys it represents the remaining quantity of food in that food source after serving all successors monkeys. While the minimum boundary of dir_{ig} represents the minimum cost (distance and time) for a spider monkey to be fed. Furthermore, $mini_g$ boundary represents the process of finding a serving VM within the minimum possible response time and the minimum possible completion time (makespan), which is the goal of the SMO-LB method (reducing response time and makespan).

So, a group of spider monkeys (UTG) should forage for the nearest ($Mini_g$) and sufficient enough (Max_g) region of food sources (VMG), that could be achieved if and only if the members of the group (UTs) are heading the best directions (dir_{ig}).

Based on the previous conclusion, a Spider Monkey (SM_i) value is the value of its current direction optimality fitness. Also, spider monkeys should be in a continuous and dynamic foraging process to find the best possible directions and to prevent a food source from being overloaded. That will be achieved through the phases of the SMO algorithm (previously stated in section 1.1).

$$SM\,(i\ ,j\ ,g\,) = SM_{mini\,ig} + U\,(0,1) + \left(SM_{max_{ig}} - SM_{mini_{ig}}\right) \quad (16)$$

SM is a 3-dimensional vector where: *i* is the SM id, *j* is the group that i^{th} SM belongs to, g is SM's current dimension id, U(0,1) is a uniformly distributed random number in the range [0,1].

- VMs Grouping Strategy: another contribution in this research paper is our adopted VMs grouping strategy, which is designed to paralyze VMs concerning the estimated traffic. We assume that a data center's flow of traffic is varying during day's hours, to handle this assumption we divide day's hours into three intervals based on the intensity of the working load. Interval 1 (working hours): *08:00 am – 04:00 pm*, Interval 2 (evening hours): *04:00 pm – 12:00 am*, and Interval 3 (night hours): *12:00 am – 08:00 am*. We expect that the flow of UTs varies decreasingly from Interval 1 to Interval 3. However, this assumption was used to figure out the best number of VMs groups for each Interval, in which those groups work in parallel mode to serve the traffic flow with minimum possible time. Measuring estimated traffic during each Interval is based on statistical information about the data center traffic flow.

The statistical information is collected from the data center and the estimated traffic during Interval *i* is computed as follows:

- Estimated traffic during Interval *i*:

UT's average length in the estimated traffic:

$$estUT_{length} = \frac{Total\ Traffic\ Length\ during\ Interval\ i}{Service\_rate\,(million\_instruction\_per\_second\,)} \quad (17)$$

The average number of UTs in the estimated traffic:

$$estUT_{.number} = \frac{Total\ Traffic\ Length\ during\ Interval\ i}{estUT_{.length}} \quad (18)$$

From Equations 17 and 18 we can estimate the *required processing capacity* per second for serving the estimated traffic:

$$reqPC_{.per\ second} = \frac{PC_{Sys}}{estUT_{.number} * estUT_{.length}} \quad (19)$$

So, VMs are grouped in which the Processing Capacity of each group should be equal to the $reqPC_{.per\ second.}$

Algorithm 2: VMs Grouping

1: *Collect statistical information form data center*
2: *for each Interval i DO:*
3: *Calculate* $estUT_{.number}$, $estUT_{.length}$, $reqPC_{.per\ second}$, *__ from equations 17,18 and 19*
4: *Maximum number of VMs groups:mg =* $PC_{Sys/reqPC.per\ second}$
5: *if mg≤ m then: # m is number of all VMs in the data center*
6: *consider each VM as a* VMG_g
7: *VMG.Int(i) ← set of VMs*
8: *mg=m;*
9: *exit(1);*
10: *else*:
11: *VMG.Int(i) ← group VMs into mg VMGs,__ where,* $PC_{VMGig} \leq reqPC_{.per\ second}$
12: *end for*
13: *if* $S_{.curtime}$ *is between 08:00 am and 04:00 pm then:*
14: *return(VMG.Int(1));*
15: *elseif* $S_{.curtime}$ *is between 04:00 pm and 12:00 am then:*
16: *return(VMG.Int(2));*
17: *else:*
18: *return(VMG.Int(3));*

- UTs Grouping Strategy: when the load balancingbegins, UTs in the frontend pool of a data center will be grabbed in terms of groups, where each UTs group represents a swarm of Spider Monkeys. Our grouping strategy of UTs assumes that the foraging territory contains enough food for all swarm members. Technically speaking, the remaining PC of the whole VMs should be enough to serve the next group of UTs.

Algorithm 3: UTs Grouping

1: *Available PC: AvalPC =* PC_{Sys} *-* $CL_{S.curtime}$
2: *While (*$UTG_{tg.length} \leq AvalPC$*) Do:*
3: *Assign UTs to* UTG_tg
4: *end while*
5: *return (*UTG_{tg}*);*

3.2. SMO-LB Algorithms

We propose the SMO-LB algorithm, its pseudo-code as follow:

Algorithm 4: SMO - LB

1: *# PREPROCESSING PHASE*
2: *VMG= VMs Grouping (Algorithm 2)*
3: *DIR: list of all directions to VMs*
4: *expand list of directions DIR;*
5: *# THE LOAD BALANCING PHASE*
6: *Call Load Balancing Decision (Algorithm 1)*
7: *Initialization:*
8: UTG_{tg}*= UTs Grouping (Algorithm 3)*
9: *SMswarms: list of all SM swarms __ #each swarm represents a group of UTs*
10: *SMS: swarm of SMs*
11: *SMG: group of SMs within SMS*
12: *MaxGr: maximum allowed number of SM groups per swarm*
13: *GlobalLeaderLimit = GTs, __ LocalLeaderLimit = LTs; # Thresholds*
14: *Initialize new swarm* SMS_c *__ where its population =sizeof (*UTG_{tg}*)*
15: *for each UT in* UTG_{tg} *Do:*
16: *assign SM use equations 14, 15 and 16*
17: *add SM to* SMS_c
18: *end for*
19: *Combine all members of* SMS_c *to form one group* SMG_1
20: *add* SMS_c *to SMswarms*
21: *GlobalLeardar of* SMS_c*= LocalLeader of__* SMG_1 *=min{SM | for all SMs in* SMG_1*}*
22: *for each swarm* SMS_c *in SMswarms Do:*
23: *While (any SM in* SMS_c *is not executed) Do:*
24: *generate new positions using Local Leader Phase*
25: *generate new positions using Global Leader Phase*
26: *update position of GlobalLeader using_ Global Leader Learning Phase*
27: *update position of LocalLeader using__ Local Leader Learning Phase*
28: *Call Local Leader Decision Phase*
29: *Call Global Leader Decision Phase*
30: *end While*
31: *end for*

3.3. Complexity Analysis

The overall computational complexity of our approach can be analyzed by examining the update/generate positions, group, and migrate/assign operations. In Algorithm 2, assuming that the number of VMs is m, then the group operation's time complexity is O(*m*). Similarly, in Algorithm 3 assuming that the number of UTs is n, then the group operation's time is O(*n*). To Algorithm 4 (the main algorithm), the time complexity of balancing the load of one SMS is composed of the following: the main loop will be repeated until n UTs SMs are executed, the time complexity for local leader phase and local leader decision phase is the same which is O(*n*×*m*), where for the global leader phase is O(*n*). The time complexity for the other SMO phases is not considered since they are composed of fewer not repeated operations. Thus we can figure out that the time complexity is O(*n*×(*n*×*m* + *n* + *n*×*m*)). Finally, the whole time complexity is O($n+m+n^2+2mn^2$).

4. Simulation Results and Evaluation

The performance of the SMO-LB algorithm has been evaluated through simulation done using CloudSim [6]. The version of the system is Intel Core i7 8th Generation processor, 1.8 GHz CPU, and 16 GB RAM running on Microsoft Windows 10 platform. Different experiments were performed with different autogenerated datasets and different parameter values.

The performance of the SMO-LB was assessed in comparison to the existing methods in CloudSim that are Round-Robin and Throttled methods. Different configuration settings for different experiments are utilized to assess the performance of the proposed methods in balancing different tasks load concerning the processing capacity of a cloud datacenter. Three different factors have been taken into consideration during the design of experiments that are: the number of tasks, the number of VMs, and the length of generated tasks. Accordingly, three different testing scenarios are designed that are light load, medium load, and heavy load scenario. Each scenario encompasses 15 experiments that vary either in the number of tasks or the number of VMs. But each scenario has a different range of task length.

1. Light task scenario: the length of tasks is ranged from 100 to 1000 instructions. Simulation results for this scenario shown that there is a slight difference in performance between the SMO-LB and Round-Robin, while both are surpassing the throttled method. Table 1 shows the average performance results for all experiments conducted in this scenario.

Table 1. Light task scenario simulation results.

Algorithm / Parameter	Throttled	Round-Robin	SMO-LB
Makespan (Sec)	40.02	12.36	8.01
Response Time (Sec)	21.56	7.81	3.28

2. Medium task scenario: the length of tasks is ranged from 1000 to 2000 instructions. The results for this scenario experiments show that the performance gap between the SMO-LB method and other tested methods starts to increase when the load becomes heavier. Table 2 shows the average results correspond to this scenario.

Table 2. Medium task scenario simulation results.

Algorithm / Parameter	Throttled	Round-Robin	SMO-LB
Makespan (Sec)	48.04	37.43	13.39
Response Time (Sec)	27.28	27.36	7.91

3. Heavy tasks: the length of tasks is long; it is ranged from 2000 to 5000 instructions. This case is the most general case that maybe happen in the real cloud with high processing requirements. The results proved that SMO-LB is very efficient whenever the workload is getting larger. Table 3 presents the reduction of the SMO-LB method for the average response times and makespan in comparison to other tested methods.

Table 3. Heavy task scenario simulation results.

Algorithm / Parameter	Throttled	Round-Robin	SMO-LB
Makespan (Sec)	71.15	56.95	29.18
Response Time (Sec)	43.63	38.87	18.0

The SMO-LB method reduced the response times of tasks and Makespan in comparison to the two other methods. The average performance results for all experimental scenarios, as illustrated in Figure 3, exhibiting a significant reduction in response times by 24.7% and 30.8% compared with Round-Robin and Throttled respectively to 10.7%. Furthermore, the SMO-LB method reduced makespan by 35.5%, and 53.0% in comparison with Round-Robin and Throttled respectively to 21.5%.

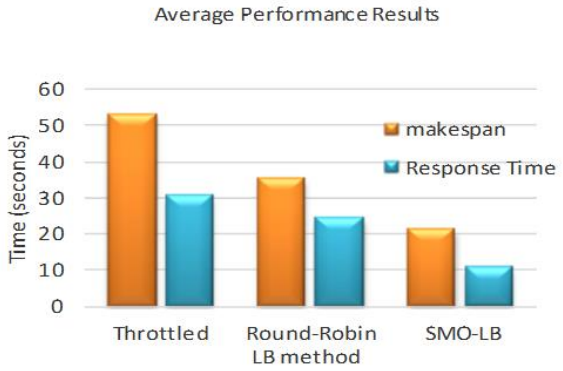

Figure 3. Overall average performance results.

We want to show that for a big number of tasks the response time and the makespan will be improved and better than other algorithms. It is proved also that Spider Monkey performs better with large values of swarms as indicated in [5].

Simulation results indicated that the SMO-LB method is efficient in keeping VMs in a balanced state for long processes, which is due to the continuous grouping and splitting of spider monkeys (user tasks) during the foraging behavior (VMs allocation).

5. Conclusions

In this paper, we have developed a load balancing method SMO-LB for cloud computing environments inspired by the foraging behavior of Spider Monkeys. To make the Spider Monkey Optimization algorithm applicable for finding the optimal load balance of tasks via virtual machines, a mathematical model was also developed for job mapping using the SMO-LB algorithm over the cloud environment.

The developed method not only handling the issue of load balancing but also takes into consideration the capability and accessibility of the resource through the proposed grouping strategies of tasks and virtual machines. The SMO-LB was tested using the CloudSim simulator with various testing scenarios and evaluated in comparison with two other existing load balancing methods, Round-Robin and Throttled.

Results have shown that the SMO-LB method is a competing method for approaching an optimal load balance state in terms of reducing task response time and makespan. Furthermore, results show that the SMO-LB method becomes more efficient whenever the load is getting heavier for available processing resources, which is due to mimicking the fission and fusion social system of spider monkeys. Future directions for this study could be carried out through an expanded investigation of performance improvement of load balancing achieved by using the SMO-LB method concerning other nature-inspired methods' improvements such as the Bee Colony or the Firefly method.

References

[1] Abunaser A. and Alshattnawi S., "Mobile Cloud Computing and other Mobile Technologies: Survey," *Journal of Mobile Multimedia*, vol. 8, no. 4, pp. 241-252, 2013.

[2] AbuNaser A., Doush I., Mansour N., and Alshattnawi S., "Underwater Image Enhancement Using Particle Swarm Optimization," *Journal of Intelligent Systems*, vol. 24, no. 1, pp. 99-115, 2015.

[3] Alla H., Alla S., Ezzati A., and Mouhsen A., "A Novel Architecture With Dynamic Queues Based on Fuzzy Logic and Particle Swarm Optimization Algorithm for Task Scheduling in Cloud Computing," *in Proceedings of International Symposium on Ubiquitous Networking*, Casablanca, pp. 205-217, 2016.

[4] Balla H., Sheng C., and Weipeng J., "Reliability-Aware: Task Scheduling in Cloud Computing Using Multi-Agent Reinforcement Learning Algorithm and Neural Fitted Q," *International Arab Journal of Information Technology*, vol. 18, no. 1, pp. 36-47, 2021.

[5] Bansal J., Sharma H., Jadon S., and Clerc M., "Spider Monkey Optimization Algorithm for Numerical Optimization," *Memetic Computing*, vol. 6, no. 1, pp. 31-47, 2014.

[6] Calheiros R., Ranjan R., De-Rose C., and Buyya R., "Cloudsim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services," *arXiv preprint arXiv*: 0903.2525, 2009.

[7] Ehsanimoghadam P. and Effatparvar M., "Load Balancing based on Bee Colony Algorithm with Partitioning of Public Clouds," *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 4, pp. 450-455, 2018.

[8] Florence A. and Shanthi V., "A Load Balancing Model Using Firefly Algorithm in Cloud Computing," *Journal of Computer Science*, vol. 10, no. 7, pp. 1156-1165, 2014.

[9] Gopinath P. and Vasudevan S., "An In-Depth Analysis and Study of Load Balancing Techniques in The Cloud Computing Environment," *Procedia Computer Science*, vol. 50, pp. 427-432, 2015.

[10] Hung P., Alam M., Nguyen H., Quan T., and Huh E., "A Dynamic Scheduling Method for Collaborated Cloud with Thick Clients," *The International Arab Journal of Information Technology*, vol. 16, no. 4, pp. 633-643, 2019.

[11] Kumar A. and Raj A., "A New Static Load Balancing Algorithm in Cloud Computing," *International Journal of Computer Applications*, vol. 132, no. 2, pp. 13-18, 2015.

[12] LD D. and Krishna P., "Honey Bee Behavior Inspired Load Balancing of Tasks In Cloud Computing Environments," *Applied Soft Computing*, vol. 13, no. 5, pp. 2292-2303, 2013.

[13] Mahmood A., Khan S., and Bahlool R., "Hard Real-Time Task Scheduling in Cloud Computing Using an Adaptive Genetic Algorithm," *Computers*, vol. 6, no. 2, pp. 15, 2017.

[14] Mansouri N., Zade B., and Javidi M., "Hybrid Task Scheduling Strategy for Cloud Computing by Modified Particle Swarm Optimization and Fuzzy Theory," *Computers and Industrial Engineering*, vol. 130, pp. 597-633, 2019.

[15] Mondal R., Nandi E., and Sarddar D., "Load Balancing Scheduling with Shortest Load First," *International Journal of Grid and Distributed Computing*, vol. 8, no. 4, pp. 171-178, 2015.

[16] Naser A. and Alshattnawi S., "An Artificial bee Colony (abc) Algorithm for Efficient Partitioning of Social Networks," *International Journal of Intelligent Information Technologies*, vol. 10, no. 4, pp. 24-39, 2014.

[17] Oktian Y., Lee S., Lee H., and Lam J., "Distributed SDN Controller System: A Survey on Design Choice," *Computer Networks*, vol. 121, pp. 100-111, 2017

[18] Pasha N., Agarwal A., and Rastogi R., "Round Robin Approach for VM Load Balancing Algorithm in Cloud Computing Environment," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 4, no. 5, pp. 34-39, 2014.

[19] Patil A., Gala H., and Kapoor J., "Dynamic Load Balancing in Cloud Computing using Swarm Intelligence Algorithms," *International Journal of Computer Applications*, vol. 130, no. 15, pp. 15-21, 2015.

[20] Patnaik S., Yang X., and Nakamatsu K., *Nature-Inspired Computing and Optimization*, Heidelberg: Springer, 2017.

[21] Rahman M., Hassan R., Ranjan R., and Buyya R., "Adaptive Workflow Scheduling for Dynamic Grid and Cloud Computing Environment," *Concurrency and Computation:*

Practice and Experience, vol. 25, no. 13, pp. 1816-1842, 2013.

[22] Singh A., Sahu S., Tiwari M., and Katare R., "Scheduling Algorithm with Load Balancing in Cloud Computing," *International Journal of Scientific Engineering and Research*, vol. 2, no. 1, pp. 38-43, 2014.

[23] Shafi U., Shah M., Wahid A., Abbasi, K., Javaid Q., Asghar M., and Haider M., "A Novel Amended Dynamic Round Robin Scheduling Algorithm for Timeshared Systems," *The International Arab Journal of Information Technology*, vol. 17, no. 1, pp. 90-98, 2020.

[24] Shukla A., Kumar S., and Singh H., "Fault Tolerance Based Load Balancing Approach for Web Resources in Cloud Environment," *The International Arab Journal of Information Technology*, vol. 17, no. 2, pp. 225-232, 2020.

[25] Snášel V., Abraham A., Krömer P., Pant M., and Muda A., "Innovations in Bio-Inspired Computing and Applications," *in Proceedings of the 6th International Conference on Innovations in Bio-Inspired Computing and Applications*, Kochi, pp. 16-18, 2015.

[26] Sreenu K. and Malempati S., "MFGMTS: Epsilon Constraint-Based Modified Fractional Grey Wolf Optimizer For Multi-Objective Task Scheduling In Cloud Computing," *IETE Journal of Research*, vol. 65, no. 2, pp. 201-215, 2019.

[27] Su S., Li J., Huang Q., Huang X., Shuang K., and Wang J., "Cost-Efficient Task Scheduling for Executing Large Programs in the Cloud," *Parallel Computing*, vol. 39, no. 4-5, pp. 177-188, 2013.

[28] Sundararaj V., "Optimal Task Assignment in Mobile Cloud Computing by Queue Based Ant-Bee Algorithm," *Wireless Personal Communications*, vol. 104, no. 1, pp. 173-197, 2019.

[29] Tawfeek M. and Elhady G., "Hybrid Algorithm Based on Swarm Intelligence Techniques for Dynamic Tasks Scheduling in Cloud Computing," *International Journal of Intelligent Systems and Applications*, vol. 8, no. 11, pp. 61-69, 2016.

[30] Xiao X., Zheng W., Xia Y., Sun X., Peng Q., and Guo Y., "A Workload-Aware VM Consolidation Method Based on Coalitional Game for Energy-Saving in Cloud," *IEEE Access*, vol. 7, pp. 80421-80430, 2019

[31] Yuan H., Bi J., Tan W., Zhou M., Li B., and Li J., "Ttsa: An Effective Scheduling Approach for Delay Bounded Tasks in Hybrid Clouds," *IEEE Transactions on Cybernetics*, vol. 47, no. 11, pp. 3658-3668, 2017.

Sawsan Alshattnawi is an Associate Professor in the Department of Computer Science at Yarmouk University (Jordan) since August 2015. She joined Yarmouk University academic staff as an assistant professor in 2009. She has received her Ph.D. degree in Computer Science from Henri Poincaré University -Nancy 1(France) in 2009, she received her B.Sc and M.Sc. degrees in computer science from Yarmouk University in 1994 and 2003, respectively. Her research interests include Distributed Systems, Cloud Computing, Mobile Computing, Internet of things, security and data science. She has been granted many research and capacity development grants.

Mohammad Al-Marie is pursuing his M.Sc. from Yarmouk University in Artificial Intelligence. He received his B.Sc. Degree in Computer Science from Zarqa University in 2005. His interest includes Optimization, Machine Learning, Computer Vision, Fuzzy Logic, and Pattern Recognition.