International Journal of Engineering & Technology, 7 (4) (2018) 3630-3637

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET
doi: 10.14419/ijet.v7i4.19777
Research paper

FOA: a meta heuristic approach for load balancing in cloud computing

Subhadarshini Mohanty [1] *, Prashanta Kumar Patra [2], Mitrabinda Ray [1]

[1] Siksha 'O' Anusandhan University, Department of Computer Science and Engineering, Bhubaneswar, India
[2] College of Engineering and Technology, Department of Computer Science and Engineering, Bhubaneswar, India
**Corresponding author E-mail:subhadarshini.mohanti@gmail.com*

Abstract

Cloud computing is a distributed computing framework that provides computational facilities, where information and assets are recovered from cloud service provider via the web, by means of a well-framed online application. But, resource sharing often leads to unavailability of resources, resulting in a deadlock situation. One approach to avoid this is by disseminating the workload uniformly between the encumbered and idle machines. This is called load balancing. The aim of doing this is to reduce the average response time and maximize resource utilization. Forest Optimization Algorithm (FOA) is based on governance of trees in the forest and survival of distinct trees. These trees have proper geological and developing conditions. The proposed Algorithm is an attempt to find these distinguished solutions from the pool to avoid starvation of tasks, at the same time improving the average response time by utilizing the process of seed dispersal in forests.

Keywords:*Average Response Time; Cloud Computing; Forest Optimization Algorithm; Load Balancing; Total Execution Time; Virtual Machines.*

1. Introduction

Cloud computing focuses on managing and providing computing services and resources online. It uses both parallel and distributed computing where hardware, software, and information are shared on demand [1]. Using these services, the customers can use these remote utilities and pay according to their usages. The virtualization technology is the key to such issues. In this model, the virtual machines (VMs) act as the execution unit fills in as the establishment for cloud computing solutions. The computational task should be assigned to the least utilized virtual machines from the dynamic array of the VMs, considering the demands of each task. The client's requests are forwarded to the most suitable data center. These requests are processed by a VM of the data center's choice. Load balancing is a pre-requisite for improved performance and efficient utilization of resources. The total processing power serves as the bottleneck for computing resources available for a VM. Thus, load balancing becomes an important factor in determining the system's overall performance. The arrival of heavy and resource intensive tasks can cause some server to be overexploited while other servers are underutilized. Symmetric distribution of the load ensures a balanced performance. Efficient scheduling of jobs and proper resource allocation are used to index the system's performance. This ensures slower cost and better average response time. Thus, it becomes important to develop an algorithm which symmetrically distributes the workload between the available virtual machines.

To adjust the solicitations of the assets it is essential to perceive a couple of significant objectives of load balancing algorithms: Prioritization of tasks: Prioritization of the tasks must be done through the calculations itself for better support of the essential and highly organized jobs despite equivalent prioritization for each of the tasks, paying little attention to their origin [2].

Flexibility and expandability of resource: The circulated framework, where the algorithm is being implemented is susceptible to change over several physical attributes. In this way, the calculations must be acceptable and sufficiently adaptable to deal with such physical changes effectively.

Cost reduction: The essential point is to accomplish a significant improvement in the execution of the framework to lower the cost incurred. In addition to these goals, the following metrics are needed to be improved for seamless customer service [3, 4].

1) Performance metrics: Performance refers to the amount of work that the system can commit over a period. If all the related constraints are optimized, then the performance of the framework may be improved.
2) Response time: In distributed frameworks, it is described as the time taken by a load balancing strategy to start processing the assigned tasks. This time ought to be small for better execution.
3) Throughput of the system: It is the aggregate of several assignments that have finished execution in each time frame. It is essential to have high throughput for better execution of the framework.
4) Make span: the processor (virtual machines) for executing the tasks the maximum time takes Make span. Suppose there are five different tasks with different burst time 1, 8, 7, 9, 10 time unit respectively. The tasks are assigned to execute in three virtual machines (VMs). Assuming that, Task t1 executes in VM1, Tasks t2, t3, t5 execute in VM2, and task t4 executes in VM3. This leads to the following completion times for different VMs i.e. VM1 = 1, VM2 = 25, and VM3 =9. Hence, the make span in above solution is 25-time unit.
5) Fault tolerance at the execution time: We can characterize it as the capacity to perform load balancing by the proper cal-

Copyright © 2018SubhadarshiniMohanty et. al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

culations without loss of connection. Each load balancing algorithm must have a decent fault tolerance.

6) Scalability of resources: Scalability is the capacity of the framework to perform load balancing with a very limited number of resources.
7) Utilization of resources: It is the parameter which gives the data about the degree towhich the asset is being utilized. For effective load balancing in the framework, thereshould be an ideal utilization of resources.
8) Task Overheads: It portrays the measure of operating cost amid the execution of load distribution scheming. It's an arrangement of developing tasks, between processes, and processors. For load balancing procedure to work efficiently, least overhead ought to be incurred.

Figure 1 provides us a brief overview of the operation of Load Balancing procedure. The clients make several requests from several machines through the machine interface. These are then put into a queue of tasks. Task manager then splits them into dependent and independent task queues and passes them on to the task scheduler. It forwards the requests to the resource manager for allocating the required system resources. Resource manager utilizes the load balancer to determine the entities that can optimally handle the requests. As a security measure, a firewall is put into use. The load balancer utilizes several computational procedures to determine the data center for processing the client's requests. The geographical conditions, total load on the data center and individual VMs are important factors for determining the suitable system resources for execution of the tasks.

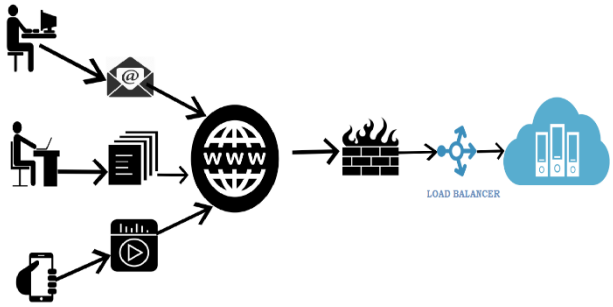


Fig. 1: Framework for Load Balancing.

FOA is enlivened by the forest in a couple of ways and reproduces the foremost evident technique of survival being seed dispersal. Those with enough daylight and nurturing environment will live longer as compared to others [5]. However most significantly, several seed dispersal techniques help them survive for an extended span of time and expand their territories [6]. Here, a Forest Optimization Algorithm for ideal load distribution is proposed, a simpler algorithm with a few control parameters. For a long time, the trees have accustomed their selves in a several ways for their survival. The plants tend to spread their seeds looking for such living spaces [7].The Forest Optimization Algorithm (FOA) is an effort to repeat such seed dispersal attributes shown in the plants.
The seed dispersal is at the core of this optimization algorithm [8]. A few seeds fall close to the parent tree and grow. This is named as local seed dispersal or "Local seeding" [9, 10]. But the majority of the seeds are diverted more distant from the parent tree by a few characteristic operators. In this way, the trees develop in various areas of the forest. This technique named as the "Global seeding". So, we use these techniques followed by the plants and trees to search for global and local optimums of the fitness function and utilized for load balancing. Our work is based on a comparative analysis of evolutionary algorithms GA, JAYA, and PSO with the Forest optimization algorithm in the cloud environment.

2. Related work

Load distribution of non-pre-emptive tasks amongst the virtual machines is at the core of task scheduling in cloud computing framework [11], [12]. It aims at maximizing throughput and minimizing response time.It avoids the situation in which a few of the resources are daintily stacked and others are over-burden, ensuring no starvation of tasks occurs in the process [13]. In distributed computing condition, the jobs arrive arbitrarily with random CPU utilization times. Because of this random arrival of jobs, specific resources are heavily loaded, while other resources remain idle. To tackle such situations, we employ space shared and time-shared modes of task execution. In the space shared mode of operation, the tasks are executed in a FIFO approach. So, each processor executes a single task and the rest of the tasks are put in a waiting queue. Therefore, the task distribution between encumbered and idle machines becomes straightforward. But, in the time-shared mode of operation, all of the tasks are executed simultaneously in parallel mode of execution. This creates intricacy in task allocation because the portion of task which has been already executed on the previous machine needs to be kept in account at real time and we need to determine the loads on each machine before making a switch. These overheads for transferring the extra load from the weighed down machines to the idle ones are significantly high. Making things more sophisticated, the cloud framework has various data centers, spread over several topographical channels. Each of them comprises of several servers known as virtual machines (VMs). At any point when a client presents an assignment, it is gathered by the cloudlets. These endeavors are dealt by data center regulator. It utilizes a VM load balancer to figure out which VM should be responsible for processing the forthcoming requests. A VM load balancer utilizes different plans to adjust the load in complex cloud computing frameworks [14]. Several load balancing algorithms have been proposed by different researchers to tackle such situations, taking such factors into their account.
Task scheduling principally comprises of a two-stage undertaking to meet the dynamic necessities of the clients [15]. This calculation accomplishes load distribution by assigning tasks to virtual machines, subsequently from the virtual machines to the host assets, respectively. It enhances the average response time of the framework. It likewise gives a better use of available resources.
Opportunistic Load Balancing is a speculation to keep every node occupied. It doesn't think about the present workload on every node [16]. The calculation is very straightforward and aides in load balancing. However, its greatest weakness is avoiding the normal execution time for some of the tasks while considering it for others. In this way, the make span becomes poor.
In Round Robin algorithm, every one of the tasks is allocated amongst the processors, just like the round-robin tournament [17]. The workload circulation is equivalent amongst the processing units. Distinctive tasks don't have a similar processing time. Sometimes a few nodes might be vigorously loaded, and others stay sit still. The time quantum decides the task designation. So, choosing a proper value for the time quantum yields a better performance[18].
Randomized algorithm is a means of static load circulation. Here, a node 'n' deals with a procedure, having a probability 'p'. At the point when every one of the jobs is of equivalent load, this algorithm functions remarkably well [19]. The issue emerges when loads are over various computational sophistications. But it isn't directed for deterministic approaches.
Min-Min Algorithm begins with an arrangement of every unassigned job and the minimum finish time for all assignments is found. From that point onward, the minimum value is chosen [20]. Formerly, the execution time for every single other job is refreshed on that piece of equipment and a similar method is rehashed until the point that every one of the undertakings is relegated to the assets. With this algorithm, the primary issue is starvation.
Max-Min algorithm, practically the equivalent of the min-min algorithm. Primary contrast is that, to start with, we should discover minimum execution time and afterward the maximum value is chosen.In the wake of finding the thoroughgoing time, the task runs on the chosen machine. At that point, the execution time for all the tasks are refreshed. This is finished by sampling the execu-

tion time of the tasks assigned, with the execution times of other tasks. At that point, all the assigned tasks are expelled from the itemize that has been executed by the framework.

Honeybee algorithm is a nature-enlivened self-association algorithm. It uses local server operations to accomplish global load distribution [21]. The execution of the framework cab be improved by an expansion in assorted variety. The significant downside of this algorithm is that throughput isn't expanded with an expansion in size of the framework.

In Active Clustering, a similar set of nodes in the framework is assembled and they cooperate. It is a self-aggregation load distribution procedure where the grid is redone to adjust the burden on the framework [22]. Framework enhancement utilizes comparative employment assignments by associating comparative administrations. Framework Performance enhances with enhanced assets. The throughput is enhanced by viable utilization of these resources.

Compare and Balance algorithm is utilized for achieving asymmetry condition and oversee uneven framework load [23]. This algorithm depends on the probability of various virtual machines running on the present host and the entire cloud framework. The present host is contrasted with the chosen one. On the off chance that a load on the current host is greater than the chose to have, it exchanges the additional load on that node. Each of the hosts plays out a similar methodology.

Lock-free multiprocessing for load distribution, proposes a lock-free multiprocessing model that maintains a strategic distance from the utilization of shared memory rather than other multiprocessing load reduction arrangements where the joint memory is utilized, and a lock is utilized to track a client session [23]. It is accomplished by adjusting the kernel. Such arrangements help in enhancing the general execution of load equalizer in a multi-core condition by sampling numerous load distribution processes on a single load distributor.

Ant colony optimization is a multi-specialist approach for troublesome combinatorial issues. Some cases utilizing this approach are TSP and QAP [24]. They were enlivened by the perception of the ant colonies. Ant's conduct is guided towards the survival of their colonies.

Load distribution issues are multifaceted and computationally immovable. It is very hard to discover the all-inclusive ideal arrangement by utilizing deterministic polynomial time algorithms. In this paper, we have put forward a Forest optimization algorithm to solve these problems in the cloud framework. The plans are assessed, and studies are made with other focused methodologies.

3. Proposed model

The Forest Optimization Algorithm (FOA) is organized into the following steps:

1) Local seeding
2) Population limitation
3) Global seeding

FOA is an evolutionary algorithm. So, it starts with a preliminary set of trees. Every tree can become a possible solution [25], [26]. A tree has some age and set of other variables associated with it. At the beginning of the procedure, the age of the tree is set to '0'. Thereafter, new trees are generated via local seeding and therefore the forest is updated with these trees. Their age is set as '0' and their parent's age is incremented by '1'. Then we enter the population control section. Some trees are skipped to create the candidate population for the subsequent stage of global seeding. In global seeding, new candidate population is taken into consideration and redundant local optimums are eliminated. Later, the trees are sorted subjected to their calculated fitness. The one with the most effective value is chosen and its age is set to '0'. It ensures that the tree doesn't age and perpetually stays within the forest. The process repeats itself until the termination criteria is met. We explain the procedure of finding the optimal solution using this algorithm following the steps described in the flowchart given below. The flowchart of the proposed model is depicted in Figure2.

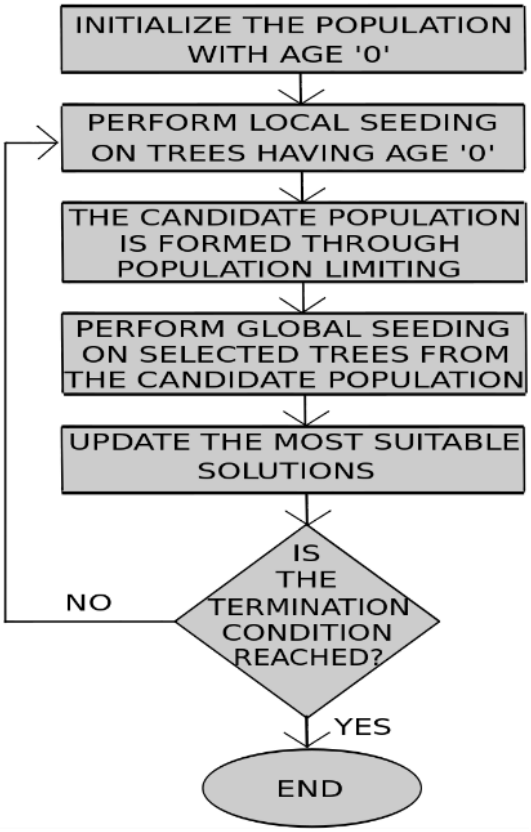


Fig. 2: Flow Chart of FOA.

Forest Optimization Algorithm for Load Balancing

Parameter preset: LSC, GSC, life time, area limit and transfer rate

1) Initialize the initial population with random solutions in the given range
2) While the termination condition is not reached,

Perform Local Seeding on the solutions of age = 0

For I = 1 to LSC

A parameter of the potential solution is randomly selected and assumed up with a small amount dx, where dx $\in$ [Ox, Δx]. The age of the potential solution is increased by a factor of 1 except for the new solutions that have been generated Perform population limiting. Remove the solutions with age greater than the specified lifetime and add them to the pool of candidate solutions. Sort the solutions based on their fitness values. Remove the solutions, which go beyond the area limit and add them to the pool of candidate solutions. Perform Global Seeding Solutions from the pool of candidate solutions are selected based on the transfer rate. For each selected solution GSC numbers of variables of the solution are selected randomly. Modify the values of the selected variables with some unevenly produced value within the given range, and add the new solution, having age = 0 to the pool of solutions. Mark the best solution thus far Sort the solutions based on their fitness value Modify the age of best tree to 0. Return the best solution.

4. Illustration of FOA for load balancing

To understand the steps of FOA for load balancing in a more lucid manner, it has been explained through an example. Let t1, t2, t3, t4, t5, t6, t7, t8, t9, t_{10} be ten different tasks with random burst time. Let these tasks be allocated to five different virtual machines such as and randomly in five different possible solution sizes of 5. To make things clearer ten tasks with 80 ms, 140 ms, 80 ms, 140 ms, 80 ms, 140 ms, 80 ms, 140 ms, 80 ms and 140 ms respectively

are assumed for computation. We aim to reach closer to the desired result where each VM is being utilized for 220 ms.

4.1. Initialization of trees

In this algorithm, the potential solutions are called trees. Each tree denotes the variable's value. Each of them has an age parameter relating to the "Age" of the corresponding tree. "Age" of recently produced trees is made equal to '0'. After local seeding phase, the ages of more seasoned trees are incremented by '1'. This incremental change in the age is later utilized for controlling the elimination of trees, beyond a certain age limit in the pool of solutions. The figure below, demonstrates a tree for a N variable, having dimensions N+1, where a_i are the estimations of the factors and the "Age" section demonstrates the changes in the age of the trees which is shown in Figure 3 [23].

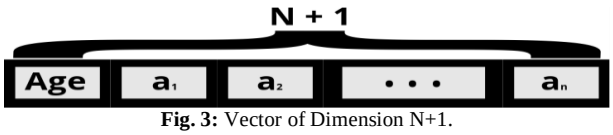


Fig. 3: Vector of Dimension N+1.

This is represented as an array of n+ 1 variable, comprising of "Age" of the tree and several other attributes. $T = [Age, a_1, a_2, ..., a_n]$ .The maximum permitted age is predetermined constraint and is denoted by "life time". This is predetermined at the beginning of the calculation. At the point where the age becomes equal to the life time, it is included in the candidate population and barred from the pool of the solution. Its value is very crucial. It shouldn't be too big or too little. If we pick a huge incentive for this parameter, in subsequent iterations of the calculation, the age is incremented, and the pool of solution is loaded with aged trees, that don't contribute to the local seeding. Then again, if we pick a little incentive for this parameter, the trees age sooner and from this time onwards they will be disregarded.For finding the solution of the problem taken as the example we set life time and initial population as 6 and 5 respectively. So, the initial population will have the tasks being assigned to 5 different VMs such that the load is distributed evenly amongst them. The 5 randomly generated possible solutions are given below in Table 1.

Table 1: Initial Population

AGE	TASK 1	TASK 2	TASK 3	TASK 4	TASK 5	TASK 6	TASK 7	TASK 8	TASK 9	TASK 10
0	4	2	4	1	0	1	1	4	4	1
0	3	0	4	3	2	0	3	2	0	3
0	3	2	4	0	2	0	3	0	0	2
0	2	0	0	1	1	2	2	2	2	4
0	3	1	1	1	0	1	3	3	2	4

4.2. Local seeding of the trees

Initially, a couple of seeds fall substantially nearer to the parent and change into infant trees. These trees face a steep competition. The fortunate ones with better nurturing conditions like adequate sunlight and better soil, transform into the victors in this front for survival. Local seeding procedure tries to emulate this technique found in the nature. It amputates those trees which have age = 0 and incorporates a few neighbours of each tree into the pool of potential solutions [27]. This is explained with an example as shown in the Figure 4. After this, ages of all the trees, besides those which are produced afterward, is incremented by 1.

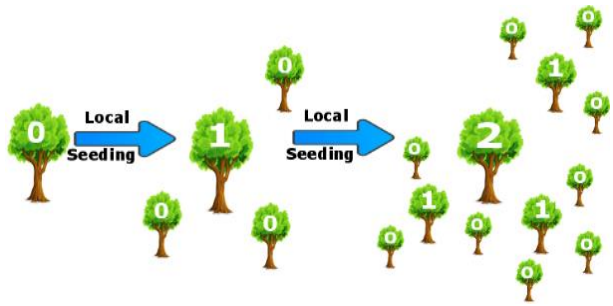


Fig. 4: Local Seeding On a Tree for a Couple of Iterations.

Incrementing the age of the trees impose a check over the number of the solutions that can be in the pool. So, if we find a promising solution, the age of that tree is reset to '0'. Then we can incorporate its neighbours into the pool of solution. Non-promising trees are routinely rejected later based on their increasing age.

Seeds dropping closer to turn into the acquaintances of the parent are treated as a constraint for this estimation. They are the so-called "Local Seeding Changes" (LSC). We have taken its value as 2 in the aforementioned example. Local seeding on a solution with age = 0 will result in 2 distinct solutions. This constraint is influenced by the extent of the problem statement and as we have a very limited search space, we have taken a minimal value as the LSC [28].

In the first cycle of the calculation, this process is applied to all the trees with age = 0. Therefore, for 'LCS' times, the number of solutions with age = 0 are added to the pool of solutions. In the subsequent iterations, the number of new trees being added will decrease because the age of the trees grows by unity and the newborn trees do not affect local seeding. Local seeding reenacts local exploration. A five-dimensional vector with LSC = 2 is explored in the example. c' and c" are two random values in the range [-Δx, Δx]. Δx is not more than the related variable's most extreme bounds. By this way, the search is confined to a particular section. With a specific end goal to play out this operation, a variable 'r' is chosen arbitrarily within the range [-Δx, Δx]. This method recurs LSC times a piece to the solutions having age = 0. We present an algebraic calculation in the figure below. We choose the values of both LSC and Δx as 1. Accordingly, a disproportionately produced value, within the array [-1, 1], is summed up with one of them which is shown in Figure 5. The new solution has age = 0 and pushed on to the pool of solutions.

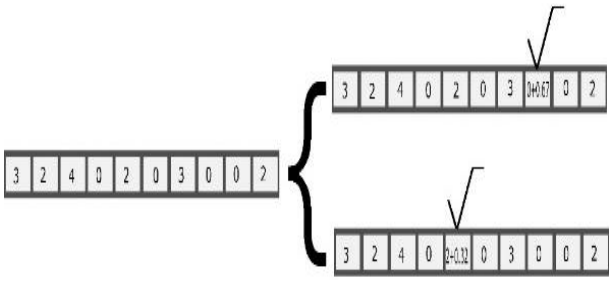


Fig. 5: Numerical Based on Local Seeding with $c' = 0.32, c' \in [-1,1]$.

Sometimes the value of the variable will be more or less similar to its upper bounds. To keep these situations at the bay, values not as much as factor's lower points of confinement and not greater than maximum cut off points are nicked from their range. So, at this stage we get 10 possible solutions from our initial population, each having age = 0. The local seeding phase is outlined in Table 2.

Since, we are adding more solutions to the set of potential solutions; we need to remove the irrelevant solutions from it. This leads us to the next phase of operation.

4.3. Population limiting

The set of potential solutions must be contained to prevent exponential growth of the set. So, we specify two parameters that restrict the size of the set. They are termed as "Life time" and "Area limit". We have set their values as 6 and 5 respectively. If the lives of the solutions, surpass the life time are expelled from the set to build the candidate population. The solutions are arranged based on their fitness values. We estimate of "area limit" parameter is same as the quantity of the initial solution so that the aggregate number stays unaltered. After population limiting the forest has the following solutions as shown in Table 3.

Table 2: Local Seeding Phase

AGE	TASK1	TASK 2	TASK 3	TASK 4	TASK 5	TASK 6	TASK 7	TASK 8	TASK 9	TASK 10
1	4	2	4	1	0	1	1	4	4	1
1	3	0	4	3	2	0	3	2	0	3
1	3	2	4	0	2	0	3	0	0	2
1	2	0	0	1	1	2	2	2	2	4
1	3	1	1	1	0	1	3	3	2	4
0	4	2	4	1	0	1	1	4	4	1
0	4	2	4	1	0	1	1	4	4	1
0	3	0	4	3	2	0	3	2	0	3
0	3	0	4	3	2	0	3	2	0	3
0	3	2	4	0	2	0	3	1	0	2
0	3	2	4	0	2	0	3	0	0	2
0	2	0	0	1	1	2	2	2	2	4
0	2	0	0	1	1	2	2	2	2	4
0	3	1	1	1	0	1	3	3	2	4
0	3	1	1	1	0	1	3	3	2	4

Table 3: Population Limiting of Initial Population

AGE	TASK 1	TASK 2	TASK 3	TASK 4	TASK 5	TASK 6	TASK 7	TASK 8	TASK 9	TASK 10	Fitness
0	3	2	4	0	2	0	3	1	0	2	0.001698
0	3	0	4	3	2	0	3	2	0	3	0.00142
0	3	0	4	3	2	0	3	2	0	3	0.00142
1	3	0	4	3	2	0	3	2	0	3	0.00142
0	3	2	4	0	2	0	3	0	0	2	0.0011

In addition, the candidate population is depicted in Table 4.

Table4: Candidate Population

AGE	TASK 1	TASK 2	TASK 3	TASK 4	TASK 5	TASK 6	TASK 7	TASK 8	TASK 9	TASK 10	Fitness
0	4	2	4	1	0	1	1	4	4	1	0.0011
0	4	2	4	1	0	1	1	4	4	1	0.0011
1	3	2	4	0	2	0	3	0	0	2	0.0011
1	4	2	4	1	0	1	1	4	4	1	0.0011
0	2	0	0	1	1	2	2	2	2	4	0.00102
0	2	0	0	1	1	2	2	2	2	4	0.00102
1	2	0	0	1	1	2	2	2	2	4	0.00102
0	3	1	1	1	0	1	3	3	2	4	8.8E-4
0	3	1	1	1	0	1	3	3	2	4	8.8E-4
1	3	1	1	1	0	1	3	3	2	4	8.8E-4

Thereafter, we are clear to proceed to the next phase of operation using both of these possible solution sets.

4.4. Global seeding of trees

Several trees have tailored quite well to their surroundings. They use different forms of agents like animals, birds, wind, and water to reach varied geographical terrains. This is often termed as seed dispersal. So, they widen their habitats. Because of this, trees reach appropriate surroundings and grow in abundance. Global seeding simulates this sort of seed distribution scheme of the trees. Global seeding is performed on a few solutions chosen from the candidate population. This is defined beforehand as the ‘‘transfer rate’’. First, a couple of the mare hand-picked from the candidate population. The values of the variables for each one is haphazardly assigned. This is aimed at considering the entire search space rather than a selected region. This leads to the addition of some new solutions to the initial population. The number of variables which can be changed is given by ‘‘Global Seeding Changes’’ or ‘‘GSC’’. Figure 6 shows the global seeding in a continuous area. We set its value as two. Hence, we choose two random variables and their values are modified to q′ and r′ inside the range of the variable.

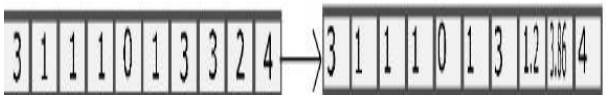


Fig. 6: Global Seeding on One Tree.

The corresponding numerical representation is shown in Table 5.

Table 5: Global Seeding Phase

AGE	TASK 1	TASK 2	TASK 3	TASK 4	TASK 5	TASK 6	TASK 7	TASK 8	TASK 9	TASK 10	Fitness
0	3	1	1	1	0	1	3	1	4	4	6.71387 E-4

Let’s assume the range to be within [-5, 5]. So, the values of the selected, two variables can be assigned values 1.7 and 0.4 which can be round off to 2 and 0 respectively.

4.5. Updating the best tree

The solutions are sorted depending upon the fitness values. The one having the maximum fitness is considered the best. Its age is set to 0. This prevents its aging during the local seeding. Hence, the best tree reaches its local optimum by local seeding. The updating of the population after global seeding phase is shown in Table 6.

Table 6: Updating the Population after Global Seeding Phase

AGE	TASK1	TASK 2	TASK 3	TASK 4	TASK 5	TASK 6	TASK 7	TASK 8	TASK 9	TASK 10	Fitness
0	3	2	4	0	2	0	3	1	0	2	0.001698
1	3	0	4	3	2	0	3	2	0	3	0.001420
0	3	0	4	3	2	0	3	2	0	3	0.001420
0	3	0	4	3	2	0	3	2	0	3	0.001420
0	3	2	4	0	2	0	3	0	0	2	0.0011
0	3	1	1	1	0	1	3	1	4	4	6.713867E-4

4.6. Termination conditions

Three different termination conditions are possible:

1) Coming to a predefined number of iterations.
2) Insignificant changes in the fitness estimation of the best tree over a few iterations.
3) Achieving a specific level of accuracy.

Out of three above conditions we have chosen the first approach i.e. to run the process 5 times and obtained the allocation which is shown in Table 7.

Table 7: Final Allocation of Tasks

TASK 1	TASK 2	TASK 3	TASK 4	TASK 5	TASK 6	TASK 7	TASK 8	TASK 9	TASK 10
3	2	4	0	2	0	4	1	1	3

This gives us a distribution of {0=280, 1=220, 2=220, 3=220, and 4=160}. This is a fair result considering that we only had 5 iterations to run and every VM gets exactly two tasks to run. The forest optimization algorithm which is used for finding out the solution is described below.

5. Simulation and result discussion

Cloudsim is open source software that simulates different aspects of a cloud framework [29]. The numerical simulation results were obtained with cloudsim-3.0.3 running on a PC having a core i7 processor 4770K clocked at 3.7 GHz, 1866 MHz 8GB LP-DDR3 RAM, running a 64-bit windows 10 operating system on512 GB of NVMe SSD. We have set life time as 6, LSC as 2, GSC as 3, area limit 30, transfer rate 10 and initial population as 30. The VM parameters are set as 512 MB memory, 10000 MB storage, 1000 Hz bandwidth and a single core CPU with 1000 MIPS. A comparative study has been done on a proposed algorithm with three other evolutionary algorithms i.e. GA, PSO, and Jaya. The comparisons are made with varying different parameters like average response time and total execution time are taken into consideration at the time of analysis. FOA for load balancing ensure significant improvements in the performance. Table 8 represents the average response time of Forest optimization, GA, PSO, and Jaya algorithms using 1000 tasks respectively.

Table 8: Comparative Analysis of Average Response Time for 1000 Tasks

VM	FOREST	GA	PSO	JAYA
50	11.1574	11.7159	11.3386	11.3782
100	6.0803	6.3469	6.2206	6.2460
200	3.6605	3.7878	3.6788	3.7693
500	2.1872	2.3952	2.1518	2.2085

Table 9 represents the total execution time of Forest optimization, GA, PSO, and JAYA algorithms using 1000 tasks respectively.

Table 9: Comparative Analysis of Total Execution Time for 1000 Tasks

VM	FOREST	GA	PSO	JAYA
50	1045.402	1050.105	1045.412	1052.23
100	1049.1615	1051.305	1049.55	1051.6399
200	1063.408	1060.49	1064.305	1060.105
500	1073.001	1072.60	1073.18	1077.13

Table 10 represents the average response time of Forest optimization, GA, PSO, and JAYA algorithms using 500 tasks respectively.

Table 10: Comparative Analysis of Average Response Time for 500 Tasks

VM	FOREST	GA	PSO	JAYA
50	5.9175	6.343	6.312	6.2
100	3.63	3.745	3.6343	3.633
200	2.592	2.56	2.386	2.364
500	2.019	1.75	1.65	1.626

Table 11 represents the total execution time of Forest optimization, GA, PSO, and JAYA algorithms using 500 tasks respectively.

Table11: Comparative Analysis of Total Execution Time for 500 Tasks

VM	FOREST	GA	PSO	JAYA
50	524.71	527.70	525.30	526.98
100	529.08	531.225	530.6725	530.392
200	533.788	536.166	536.78	536.89
500	538.9	540.58	538.37	540.28

Table 12 represents the average response time of Forest optimization, GA, PSO, and JAYA algorithms using 100 tasks respectively.

Table12: Comparative Analysis of Average Response Time for 100 Tasks

Vm	Forest	Ga	Pso	Jaya
50	2.4825	2.2189	2.0803	2.015
100	2.02	1.6815	1.60088	1.6315
200	1.5622	1.379	1.2427	1.2887
500	1.4958	1.33	1.2182	1.257

Table 13 represents the total execution time of Forest optimization, GA, PSO, and JAYA algorithms using 100 tasks respectively.

Table13: Comparative Analysis of Total Execution Time for 100 Tasks

VM	FOREST	GA	PSO	JAYA
50	107.527	107.388	107.466	107.637
100	107.486	107.698	107.823	107.712
200	107.893	108.073	108.943	108.6
500	107.421	107.664	107.358	107.328

From the above result analysis it is observed that the proposed Forest Optimization Algorithm performs better than GA, PSO, and JAYA when the tasks to VM ratio are significantly high. The graphs are drawn considering average response time for 1000, 500, and 100 tasks in Figure 7, Figure 8, and Figure 9 respectively.

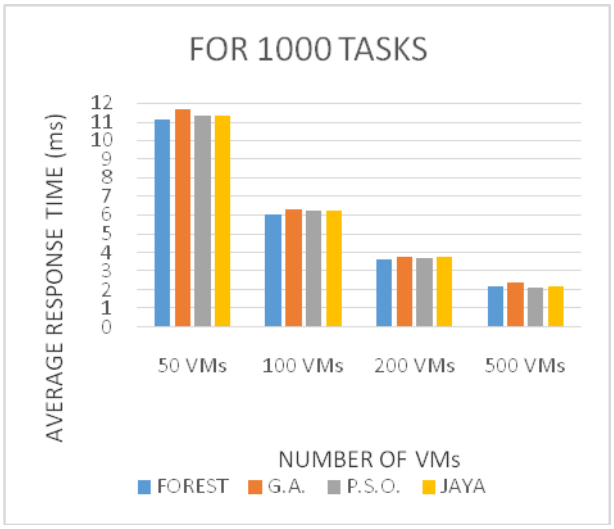


Fig. 7: Average Response Time (Ms) for 1000 Tasks.

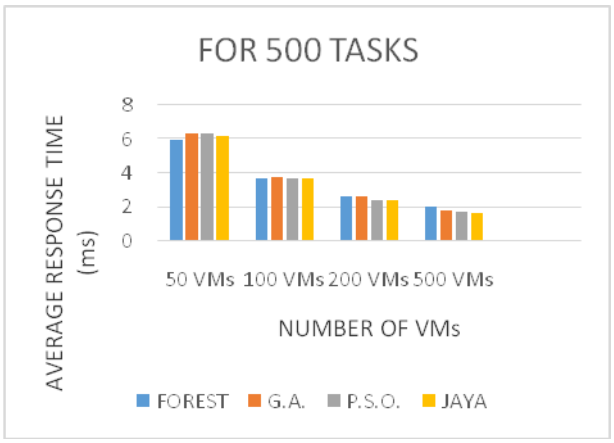


Fig. 8: Average Response Time (MS) for 500 Tasks.

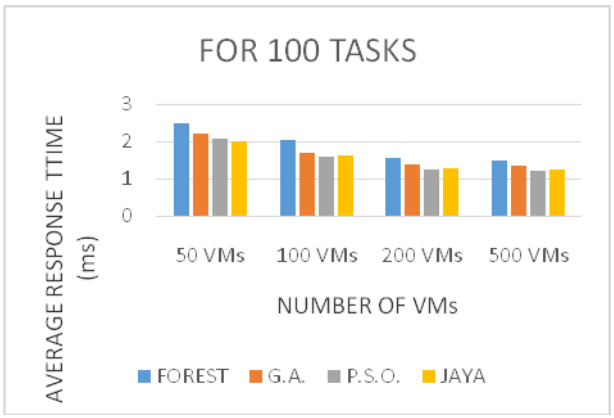


Fig. 9: Average Response Time (MS) for 100 Tasks.

The graphs are drawn considering total execution times for 1000, 500, and 100 tasks and shown in Figure 10, Figure 11, and Figure 12 respectively.

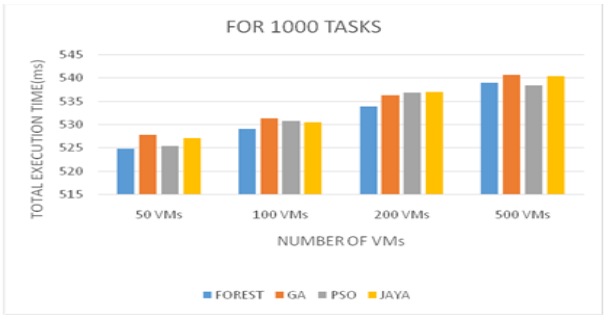


Fig. 10: Total Execution Time (MS) for 1000 Tasks.

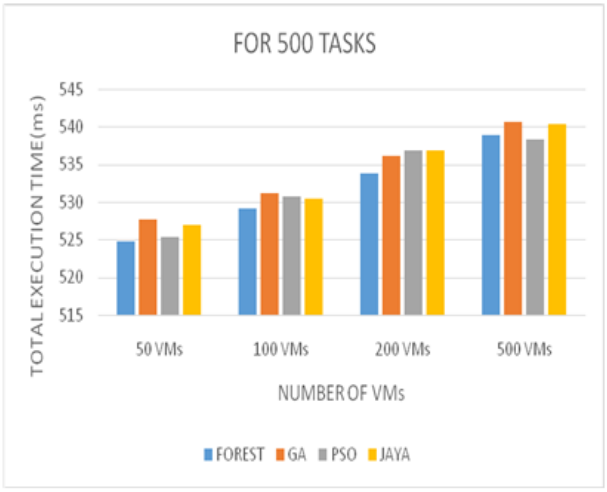


Fig. 11: Total Execution Time (MS) For 500 Tasks.

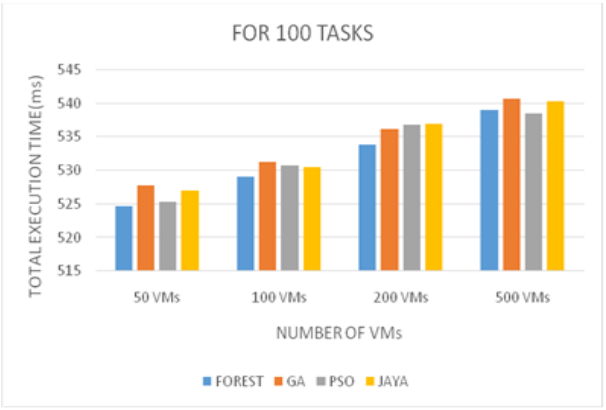


Fig. 12: Total Execution Time (MS) for 100 Tasks.

6. Conclusion

A comparison has been made with GA, PSO, and JAYA by conducting experiments to grasp the performance of the algorithms. The simulation result shows that the proposed FOA algorithmic outperforms GA, PSO, and JAYA by minimizing the average response and total execution time under significant load. If the ratio of total number of tasks to that of total number of VMs is considerably high, we see drastic performance improvement using the FOA. However, as the ratio decreases in worth, the performance degrades. But the parameters can be tweaked to perform even better under lower loads. So, as the future work, we aim at finding appropriate values of these system parameters for a balanced performance under heavy, light and moderate loads.

References

[1] Xiao Z, Song W, Chen Q. Dynamic resource allocation using virtual machines for cloud computing environment. IEEE transactions on parallel and distributed systems. Vol. 24, No. 6, (2013), pp.1107-1117.https://doi.org/10.1109/TPDS.2012.283.

[2] Chaczko Z, Mahadevan V, Aslanzadeh S, Mcdermid C. Availability and load balancing in cloud computing. In the proceedings of the International Conference on Computer and Software Modeling, Singapore ,Vol. 14, (2011).

[3] Foster I, Zhao Y, Raicu I, Lu S. Cloud computing and grid computing 360-degree compared. In the proceedings of the workshop on Grid Computing Environments, (2008), pp.1-10.

[4] Buyya R, Ranjan R, Calheiros RN. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In the proceedings of the International Conference on Algorithms and Architectures for Parallel Processing, (2010). Pp.13-31.

[5] Hubbell SP. Tree dispersion, abundance, and diversity in a tropical dry forest. Vol. 203, No. 4387, (1979), pp.1299-309.
[6] Zhu, J., Shan, Y., Mao, J. C., Yu, D., Rahmanian, H., & Zhang, Y. (2017). Deep embedding forest: Forest-based serving with deep embedding features. In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and DataMining (pp. 1703-1711).https://doi.org/10.1145/3097983.3098059.
[7] Howe HF, Smallwood J. Ecology of seed dispersal. Annual review of ecology and systematic. Vol. 13, No. 1, (1982), pp.201-28.https://doi.org/10.1146/annurev.es.13.110182.001221.
[8] Chen, J., Li, K., Tang, Z., Bilal, K., Yu, S., Weng, C., & Li, K. A parallel random forest algorithm for big data in a spark cloud-computing environment. IEEE Transactions on Parallel & Distributed Systems, (2017).https://doi.org/10.1109/TPDS.2016.2603511.
[9] Green DS. The efficacy of dispersal in relation to safe site density. Oecologia. Vol 56, No. 2, (1983), pp.356-8.https://doi.org/10.1007/BF00379712.
[10] Panda, Sanjaya K., and Prasanta K. Jana. "Normalization-based task scheduling algorithms for heterogeneous multi-cloud environment." Information Systems Frontiers, Vol. 20, No. 2, (2018), pp. 373-399.https://doi.org/10.1007/s10796-016-9683-5.
[11] Panda, S., Nanda, S., &Bhoi, S. A pair-based task-scheduling algorithm for cloud computing environment. Journal of King Saud University – Computer and Information Sciences, (2018).
[12] Cain ML, Milligan BG, Strand AE. Long distance seed dispersal in plant populations. American Journal of Botany. Vol-87, No.9, (2000), pp.1217-1227.https://doi.org/10.2307/2656714.
[13] Ozşen S, Güneş S. Attribute weighting via genetic algorithms for attribute weighted artificial immune system (AWAIS) and its application to heart disease and liver disorders problems. Expert Systems with Applications. Vol. 36, No.1, (2009), pp.386-392.https://doi.org/10.1016/j.eswa.2007.09.063.
[14] Desai T, Prajapati J. A survey of various load balancing techniques and challenges in cloud computing. In the proceedings of the International Journal of Scientific & Technology Research. Vol. 2, No. 11, (2013), pp.158-161.
[15] Shaw SB, Singh AK. A survey on scheduling and load balancing techniques in cloud computing environment. In the proceedings of the Computer and Communication Technology ICCCT, (2014), pp. 87-95
[16] Hu J, Gu J, Sun G, Zhao T. A scheduling strategy on load balancing of virtual machine resources in cloud computing environment. In the proceedings of the Parallel Architectures, Algorithms and Programming PAAP, (2010), pp. 89-96
[17] Wang SC, Yan KQ, Liao WP, Wang SS. Towards a load balancing in a three-level cloud computing network. In the proceedings of the Computer Science and information technology ICCSIT, (2010), pp.108-113.
[18] Nusrat Pasha D, Agarwal A, Rastogi R. Round robin approach for VM load balancing algorithm in cloud computing environment. International Journal, (2014), pp.4-5.
[19] Mahmoudi, S., &Lailypour, C. A discrete binary version of the Forest Optimization Algorithm. International Academic Institute for Science and Technology, Vol. 02, No.12, (2014), pp. 10-23.
[20] Chen H, Wang F, Helian N, Akanmu G. User-priority guided Min-Min scheduling algorithm for load balancing in cloud computing. In the proceedings of the national conference on Parallel computing technologies (PARCOMPTECH), (2013), pp. 1-8.
[21] Ghafari SM, Fazeli M, Patooghy A, Rikhtechi L. Bee-MMT: A load balancing method for power consumption management in cloud computing. In the proceedings of the Sixth International Conference on Contemporary Computing (IC3), (2013), pp. 76-80.
[22] Randles M, Lamb D, Taleb-Bendiab A. A comparative study into distributed load balancing algorithms for cloud computing. In the proceedings of the 24th International Conference on Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE, (2010), pp. 551-556.
[23] Zhao Y, Huang W. Adaptive distributed load-balancing algorithm based on live migration of virtual machines in cloud. In the proceedings of the Fifth International Joint Conference on INC, IMS and IDC, 2009. NCM'09,(2009), pp. 170-175.
[24] Nishant K, Sharma P, Krishna V, Gupta C, Singh KP, Rastogi R. Load balancing of nodes in cloud using ant colony optimization. In the proceedings of the 14th International Conference on Computer Modelling and Simulation (UKSim) UKSim, (2012), pp. 3-8.
[25] Wu, Q. A Novel Optimization Algorithm: The Forest Algorithm. In the proceedings of the Fifth International Conference on Intelligent Systems Design and Engineering Applications (ISDEA), (2014), (2014), pp. 59-63.
[26] Mohialdeen IA. Comparative study of scheduling algorithms in cloud computing environment. In the proceedings of the Journal of Computer Science, (2013), pp. 252-263.
[27] Ghaemi, M., &Feizi-Derakhshi, M. R. Forest optimization algorithm. Expert Systems with Applications, Vol. 41, No. 15, (2014), pp. 6676-6687.https://doi.org/10.1016/j.eswa.2014.05.009.
[28] Ghaemi, M., &Feizi-Derakhshi, M. R. Feature selection using forest optimization algorithm. Pattern Recognition, Vol. 60, (2016), pp. 121-129.https://doi.org/10.1016/j.patcog.2016.05.012.
[29] 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, Vol. 41, no. 1, (2011), pp. 23-50.https://doi.org/10.1002/spe.995.