Materials Today: Proceedings xxx (xxxx) xxx

Contents lists available at ScienceDirect

Materials Today: Proceedings

journal homepage: www.elsevier.com/locate/matpr

An evolutionary approach to load balancing using forest optimization algorithm

Ishan Aryendu [a,*], Subasish Mohapatra [a], Subhadarshini Mohanty [a], Sachi Nandan Mohanty [b]

[a] *Department of Computer Science and Engineering College of Engineering and Technology Bhubaneswar, Odisha, India*
[b] *Department of Computer & Science Engineering, ICFAI Founda-tion for Higher Education, IcfaiTech, Hyderabad, India*

ARTICLE INFO

Article history:
Received 18 September 2020
Accepted 25 October 2020
Available online xxxx

Keywords:
Load balancing
Cloud computing
Forest optimization algorithm
Average response time
Total execution time

ABSTRACT

Cloud computing can be described as a distributed computing framework that provides computational facilities, where information and assets are recovered from cloud service providers 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 developing conditions. The proposed Algorithm is an attempt to find these distinguished solutions from the pool to avoid starvation of the tasks, at the same time improving the average response time by utilizing the process of seed dispersal in forests.

1. Introduction

Cloud computing focuses on managing and providing computing services online. It uses both parallel and distributed computing where hardware, software, and information are shared on demand [1]. Using these services, the customers can access these remote utilities and pay according to their usages. Virtualization technology is the key to such issues. In this model, the virtual machines (VMs) act as the execution unit that fill 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 prerequisite for improved performance and efficient utilization of resources. The total processing power serves as the bottleneck for computing resources available for a VM. But job arrival is unforeseeable and so are the capacities of each virtual machine. 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 servers to be over utilized 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 lower cost and better average response time. Thus, it becomes important to develop an algorithm which symmetrically distributes the workload among the available virtual machines.

To adjust the solicitations of the assets, it is essential to perceive a couple of significant objectives [2] of load balancing algorithms:

1. Prioritization of tasks: Prioritization of the tasks must be done through the calculations itself for better support of essential and highly organized jobs, despite equivalent prioritization for each of the tasks, as well as, paying little attention to their origin.
2. Flexibility and expandability of resources: 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.
3. Cost reduction: The essential point is to accomplish a significant improvement in the execution of the framework to lower the cost incurred.

* Corresponding author.
E-mail addresses: aryenduishan@gmail.com (I. Aryendu), smohapatra@cet.edu.in (S. Mohapatra), sdmohantycse@cet.edu.in (S. Mohanty).

https://doi.org/10.1016/j.matpr.2020.10.750
2214-7853/© 2020 Elsevier Ltd. All rights reserved.
Selection and peer-review under responsibility of the scientific committee of the Emerging Trends in Materials Science, Technology and Engineering.

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 [5]. It is essential to have high throughput for better execution of the framework.

Some additional properties for better load balancing are,

1. Fault tolerance at the execution time: We can characterize it as the capacity to perform load balancing by the proper calculations without loss of connection. Each load balancing algorithm must have a decent fault tolerance.
2. Scalability of resources: Scalability is the capacity of the framework to perform load balancing with a very limited number of resources.
3. Utilization of resources: It is the parameter which gives the data about the degree to which the asset is being utilized. For effective load balancing in the framework, there should be an ideal utilization of resources.
4. Task Overheads: It portrays the measure of operating costs 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.

Fig. 1 provides 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 then 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.

Here, 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 themselves in several ways for their survival [6]. No matter what, they are certain to wither and perish. Along these lines, the new trees replace the old ones. Still, a few trees pull through for a longer span of time [7]. This is a result of the nurturing natural surroundings. In this way, plants tend to spread their seeds looking for such living spaces [8]. The Forest Optimization Algorithm (FOA) is an effort to repeat such seed dispersal attributes shown in the plants.

The seed dispersal is the crucial part of this optimization algorithm. A couple of seeds fall closer to the tree and start to grow. This is named as local seed dispersal [9,10] or ”Local seeding”. Regardless, a clear majority of the seeds are redirected to a far off location from the parent tree. In this way, the trees develop in varied regions of the forest. This technique is named as the ”Global seeding”.

Our work is based on a comparative analysis of Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Flower Pollination (FPA) and Jaya optimization algorithm with the Forest Optimization Algorithm (FOA) in the cloud environment.

2. Related work

Using Genetic algorithms (GA), we iterate through a set of random solutions, based on Darwin's principles of natural selection, and optimize some user defined functions. Fortunately, it doesn't require the differentiability for optimization unlike many derivative-based optimization algorithms. So we can handle a wider range of functions. To begin with, we calculate the fitness of the initial population. The individuals are chosen from the initial population based on their fitness score over several iterations, in a process called 'reproduction'. It ensures that the individuals with better fitness-score have a better chance of being selected.

Moreover, the individuals are hybridized by using 'genetic operators'. In this process the solutions are changed locally and combined by using crossover and mutation with a probability of 0.6–0.8 and 0.01–0.001 respectively [11]. This can provide improved results over several iterations and is in line with the concept of 'generations' in evolutionary terms. We can terminate this process either by specifying the maximum iterations or some flag value for improvement in the solution's fitness. As it explores the entire pop-

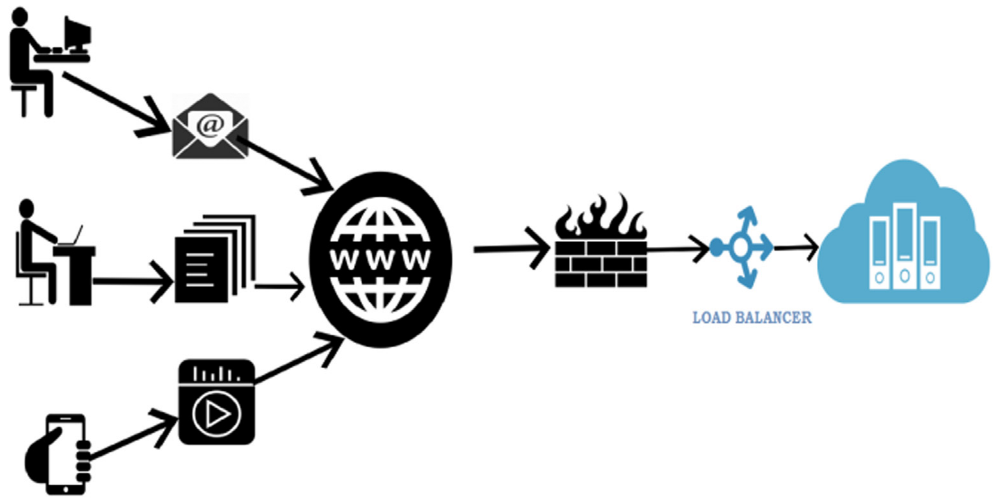

Fig. 1. Framework for load balancing.

ulation set, it has a better chance of finding the global optimum values.

Particle swarm optimization (PSO) takes its inspiration from the social dynamics of living beings. It is based on the theory that the living beings adapt to their environment, seek food, mates and protection from predators. If they are randomly scattered, they automatically adapt to their surroundings [12]. We start with a selected group of candidate population, called the swarm, containing the potential solutions, called particles. These particles are subjected to a fitness function and their arrangement is governed by their current position and the position of the best solution in the entire population.

We can take a group of birds as an example [13]. Assuming the birds as a swarm, when they are ravenous, they scan for nourishment, and can be treated as tasks. But there is a limited amount of food available in their territory, which can be related with resources. Since there are many tasks and a limited amount of resources, it simulates the condition in cloud computing environments. The birds don't have the foggiest idea where the food is to be found. In such a situation, an algorithm to discover the food source must be designed. On the off chance that each bird will attempt to discover for nourishment all alone, it may expend a lot of time. But by observing the behavior of the birds, it was found that the best way to deal with this was to pursue the birds closest to the food source. This behavior of birds is mimicked in our calculations and the algorithm so structured is named as Particle Swarm Optimization algorithm (PSO).

Flower pollination is the process of transfer of pollen through some natural agents like water, wind, insects and animals. So the pollination process can be classified as biotic and abiotic. The biotic pollination comprises about ninety percent of the total pollination and the rest falls under abiotic pollination. The flowers often attract pollinators by several properties like their color, smell, etc. These pollinators tend to prefer some flowers over the others and thus form a flower constancy. This ensures maximized reproduction [14].

Furthermore, we can classify the pollination process into self-pollination and cross-pollination. In self-pollination, the pollen is transferred from the male part of the flower to the female part of the same flower. Whereas in cross-pollination, the pollen is transferred from the male part of one flower to the female part of another flower. This can happen over a large geographical distance. Therefore, the biotic and abiotic factors play a significant role in it. The biotic pollinators like flies and bats are called global pollinators. Since the movement of these creatures is described by Levy flight, we can safely assume that it is governed by the Levy distribution [14].

To sum up the process, the biotic or the cross-pollination is termed as global pollination. The movement of these pollinators is governed by the Levy distribution. Abiotic pollination or self-pollination is treated as local pollination. The flower constancy property is represented by the reproduction ratio which is directly proportional to the relative level of comparability between two flowers. Because the flowers are physically closer to one another, local pollination is preferred over global pollination. This is constrained by the value of a probability 'p' [14]. Depending on the value of 'p' local or global pollination takes place.

Teaching-learning-based optimization (TLBO) is a population-based optimization algorithm which tries to mimic the traditional teacher and student relationship [15]. The initial population is considered as a group of students and the fitness represents the grades that they've scored. The teaching factor T_F defines the course of action to be taken for each iteration. Hence, this parameter must be tweaked in the subsequent iterations to ensure diversification of the solutions and often gets trapped in local optima. Jaya algorithm draws its inspiration from TBLO algorithm. But unlike TBLO algorithm, Jaya algorithm is easy to execute as it does not require tuning of any calculation specific parameters. We start with an initial population of randomly generated solutions within the confines of the upper and lower bounds of the variable. In the next step, we calculate the fitness value of each candidate in our population set and mark the candidates with the best and worst fitness. These are then used to modify the other candidates according to the following eq. [16].

$A(i + 1, j, k) = A(i, j, k) + r(i, j, 1)(A(i, j, b) - |A(i, j, k)|) - r(i, j, 2)(A(i, j, w) - |A(i, j, k)|)$where b, w, r (i, j, 1) and r (i, j, 2) are the best, the worst solutions and two randomly generated probability values respectively. They diversify the population and ensure proper scaling by pushing the candidates towards the best possible fitness value. Once the calculation is complete for all of the candidates, their values are compared to find the best and worst fitness for the next iteration. Therefore, the candidates move closer to the best fitness value possible with the subsequent generations.

3. Problem statement

Distributed computing conditions comprise of a few conveyed hubs interconnected by rapid connections. These hubs oversee executing attributes of applications having numerous assets and computational prerequisites. Cloud frameworks must have appropriate presets to meet the computational requests of extensive, and diverse workload.

A cloud framework, comprising of 'n' independent nodes [17] can be represented as,

$M = \{M_1, M_2, M_3, \ldots, M_n\}$

An array of 'm' different tasks can be represented as,

$J = \{J_1, J_2, J_3, \ldots, J_m\}$

Each task Ji runs on node M_j, and is represented as 'J_{ij}. 'J_{ij} 'is the expected computational time of i'th task on the j'th machine. Suppose, 'G' represents the array of tasks allotted to node M_j and the time taken by the machine M_j to finish the task be J_j. Our intention is to minimize the average response time of the framework. Suppose P_{ij} represents a pair of nodes, such that, $M_j \in M$ and task $J_{ij} \in T$ and:

(a) $P_{ij} = 0$; means that node j and task i are independent of each other.

(b) $P_{ij} = J_{ij}$; M_j processes task i.

(c) If we take L as the total load on 'm' nodes then, $\sum_{j=1}^{m} P_{ij} = J_{ij}$, for all $J_i \in T$.

(d) $\sum^{n}{}_{i=1} P_{ij} \le L$, for all $M_j \in M$.

Keeping these constraints in sight, we specify the utilization of a node as, $U = J_i$ / makespan [18] and the fitness function as, Fitness, i = U/ (n $*$ makespan) [19]. This is used as the basis for finding out the minimum response time of the system. Finally, the comparison is done amongst the optimization algorithm to determine the best performer in terms of lower average response time and total execution time.

4. Proposed algorithm

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

Local seeding.

Population limitation.

Global seeding.

FOA is an evolutionary algorithm [20] which starts with a preliminary set of trees. Every tree can become a possible solution. 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 in Fig. 2. To make things clearer we take an example where we have '5′ virtual machines and '10′ tasks with 80 ms, 140 ms, 80 ms, 140 ms, 80 ms, 140 ms, 80 ms, 140 ms, 80 ms and 140 ms respectively. We aim to reach close 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 have 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. Fig. 3, shows a tree for an N variable, having dimensions N + 1, where the estimations of the factors and the "Age" section demonstrate the changes in the age of the trees.

This is represented as an array of n + 1 variables, comprising of "Age" of the tree and several other attributes i.e., 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

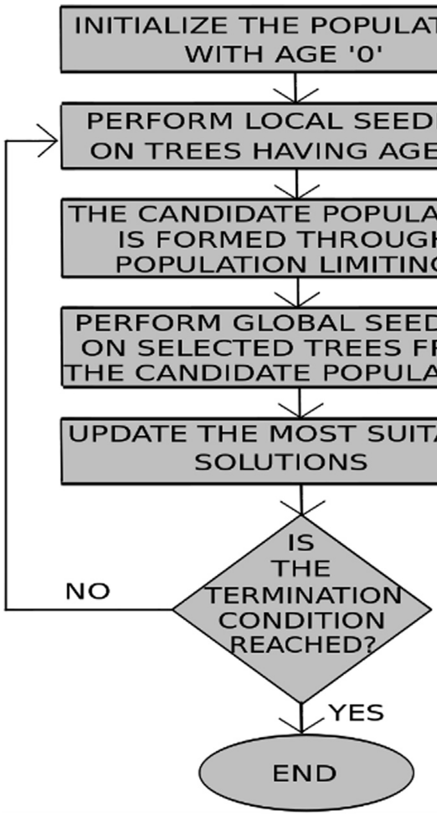
INITIALIZE THE POPULATION WITH AGE '0'
PERFORM LOCAL SEEDING ON TREES HAVING AGE '0'
THE CANDIDATE POPULATION IS FORMED THROUGH POPULATION LIMITING
PERFORM GLOBAL SEEDING ON SELECTED TREES FROM THE CANDIDATE POPULATION
UPDATE THE MOST SUITABLE SOLUTIONS
IS THE TERMINATION CONDITION REACHED?
NO
YES
END

Fig. 2. Flowchart of FOA.

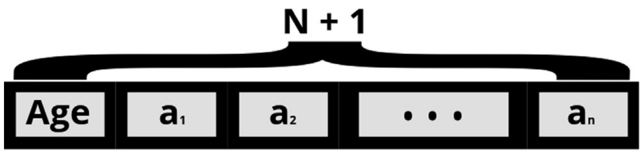

Fig. 3. Vector of dimension N + 1.

the calculation. At the point where the age becomes equal to the lifetime, 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 solutions 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 in Table 1,

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 nature. It amputates those trees which have age = 0 and incorporates a few neighbors of each tree into the pool of potential solutions. This is explained with an example as shown in Fig. 4. After this, ages of all the trees, besides those which are produced afterward, is incremented by 1.

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 neighbors 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.

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' (Fig. 5) 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 chose the values of both LSC and Δx as 1. Accordingly, a disproportionately produced value, within the array [-1, 1], is summed up with

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

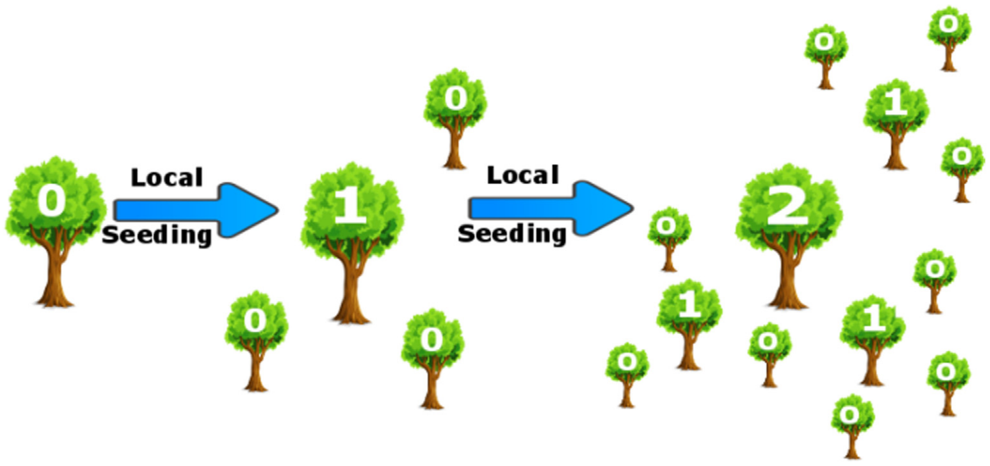

Fig. 4. Local seeding on a tree for a couple of iterations.

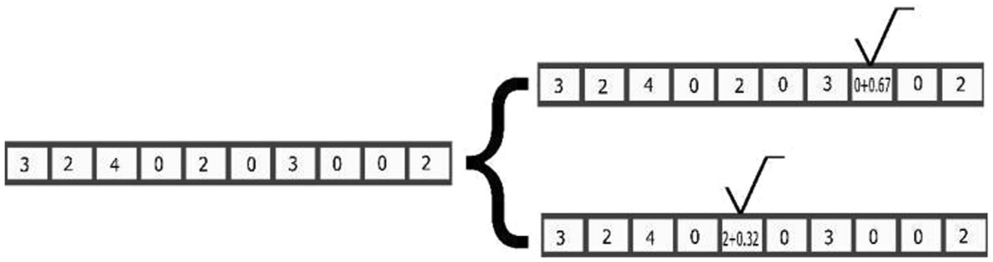

Fig. 5. Numerical on local seeding with c' = 0.32, c' ε [-1, 1].

one of them. The new solution has age = 0 and pushed on to the pool of solutions.

Sometimes the value of the variable will be more or less similar to its upper bounds. To keep these situations at bay, values not as much as factor's lower points of confinement and not greater than maximum cutoff points are nicked from their range. So, at this stage we get 10 possible solutions from our initial population, each having age = 0. These are shown 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 age of the solutions surpasses the lifetime, they are expelled from the set to build the candidate population. The solutions are arranged based on their fitness values. We estimate the ''area limit'' parameter is the same as the quantity of the initial solution so that the aggregate number stays unaltered.

After population limiting the forest has the following solutions in Table 3,

And the change in the candidate population is shown in Table 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 the 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 them are hand-picked from the candidate population. The values of the variables for each one is assigned haphazardly. This is done to consider the entire search space rather than a selected region which 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".

Table 2
Local seeding phase.

AGE	TASK 1	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

Table 4
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

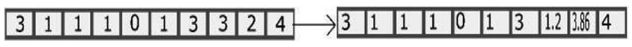

Fig. 6. Global seeding on one tree.

Fig. 6 shows the global seeding in a continuous area. We set its value as two. Hence, we choose two random variables 3 and 2 and their values are modified to 1.2 and 3.86 inside the range of the variable.

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

Table 6
Updating the population after 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	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

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

Table 5 shows the resultant solution after the global seeding phase Table 6 Table 7.

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 rounded 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.

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.

We have chosen the first approach i.e. to run the process 5 times and obtained:

This gives us a distribution of {0 = 280, 1 = 220, 2 = 220, 3 = 220, 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. Forest optimization algorithm

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

Initialize the initial population with random solutions in the given range.

While the termination condition is not reached,

Perform Local Seeding on the solutions of age = 0.

For I = 1 to LSC.

Table 8
Average response time and total execution time for different values of LSC.

LSC	2	3	4	5	6	7	8
Average Response Time	51.090	51.061	51.025	51.0157	50.944	51.06	51.185
Total Execution time	1005.15	1005.014	1005.09	1005.07	1005.05	1005.10	1005.21

Table 9
Average response time and total execution time for different values of GSC.

GSC	1	2	3	4	5	6	7
Average Response Time	51.097	51.090	51.02	51.017	51.012	50.988	51.064
Total Execution time	1005.156	1005.151	1005.145	1005.137	1005.031	1005.024	1005.358

Table 10
Average response time and total execution time for different values of Transfer Rate.

Transfer Rate	5	10	15	20
Average Response Time	51.16	51.027	51.034	51.04
Total Execution time	1005.16	1005.09	1005.107	1005.112

Table 11
Comparative analysis of average response time for 1000 tasks.

VM	GA	PSO	JAYA	FPA	FOA
50	11.4	11.141	11.144	11.225	11.059
100	6.28	6.164	6.101	6.058	6.029
1000	1.634	1.573	1.571	1.573	1.571

Table 12
Comparative analysis of total execution time for 1000 tasks.

VM	GA	PSO	JAYA	FPA	FOA
50	1015.049	1014.995	1014.87	1015.542	1015.363
100	1022.243	1022.624	1022.395	1021.844	1023.313
1000	1034.272	1034.574	1035.079	1035.175	1033.921

Table 13
Comparative analysis of average response time for 10,000 tasks.

VM	GA	PSO	JAYA	FPA	FOA
50	101.797	101.464	101.398	101.381	101.366
100	51.782	51.371	51.323	51.298	51.235
1000	6.368	6.162	6.151	6.155	6.152

Table 14
Comparative analysis of total execution time for 10,000 tasks.

VM	GA	PSO	JAYA	FPA	FOA
50	10041.172	10041.036	10041.152	10040.720	10041.083
100	10055.393	10055.515	10055.623	10055.550	10055.655
1000	10225.272	10228.183	10228.271	10226.085	10229.729

Table 15
Comparative analysis of average response time for 100,000 tasks.

VM	GA	PSO	JAYA	FPA	FOA
50	1003.818	1003.808	1003.813	1003.812	1003.787
100	502.553	502.551	502.548	502.549	502.516
1000	51.549	51.413	51.388	51.473	51.403

Table 16
Comparative analysis of total execution time for 100,000 tasks.

VM	GA	PSO	JAYA	FPA	FOA
50	100275.287	100274.730	100274.742	100274.708	100274.504
100	100291.667	100291.841	100292.489	100290.584	100291.655
1000	100558.563	100559.517	100559.313	100559.970	100559.676

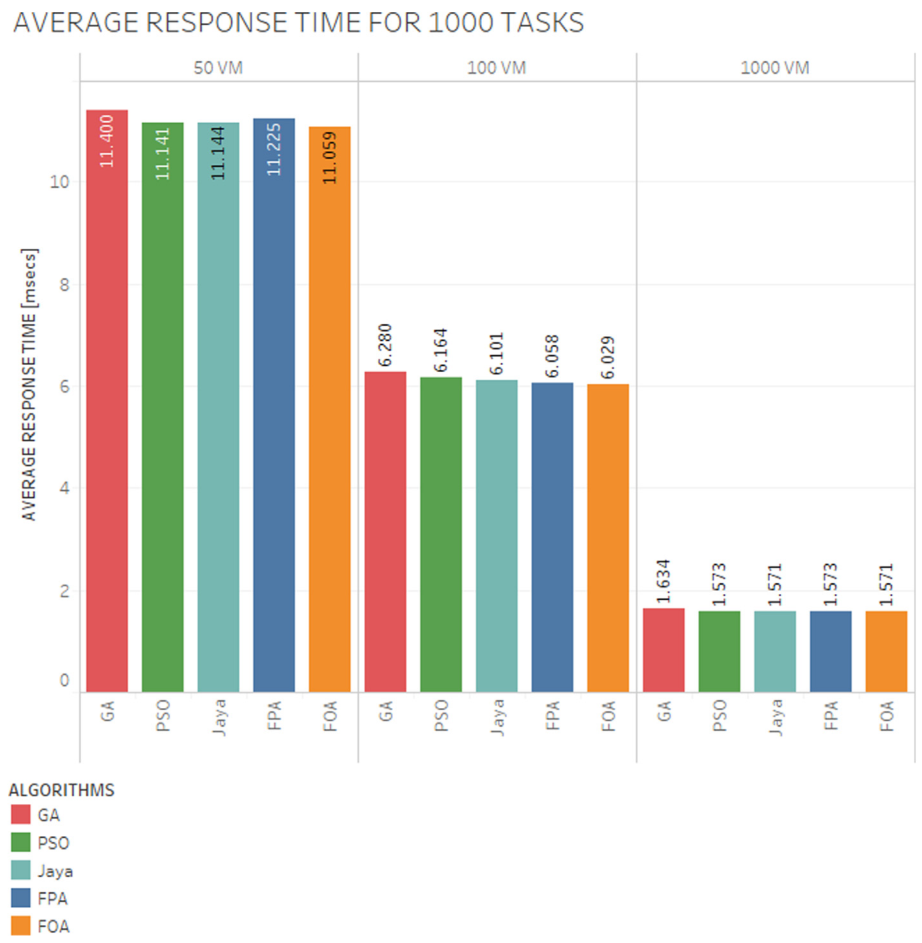

Fig. 7. Average response time (ms) for 1000 tasks and (a) 50VMs (b) 100VMs (c) 1000 VMs.

A parameter of the potential solution is randomly selected and summed up with a small amount dx, where dx [-Δx, 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 life time 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 number 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.

6. Observations and graphs

Cloudsim is open source software that simulates different aspects of a cloud framework. The numerical simulation results were obtained with cloudsim-3.0.3 running on a PC having a core i7 processor 4770 K clocked at 3.7 GHz, 8 GB of LP-DDR3 1866 MHz RAM, running a 64-bit Debian Buster operating system on 512 GB of NVMe SSD. In order to determine the optimal values for the parameter presets, we have calculated the values of average response time and total execution time, using 10 VMs and 1000 tasks, for different values of LSC, GSC and transfer rate.

Table 8 shows the average response time and total execution time for several values of LSC ranging from 2 to 8, keeping GSC constant at 2.

Table 9 shows the average response time and total execution time for several values of GSC ranging from 1 to 7, keeping LSC constant at 2.

From Table 8 and Table 9, we get the best values of average response time and total execution time for LSC ‘6′ and GSC ‘6′. Using these values, we calculate the best value for transfer rate under heavy load.

Table 10 shows the average response time and total execution time for several values of Transfer Rate, keeping both LSC and GSC constant at 6.

We have set life time as 6, LSC as 6, GSC as 6, 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. Also, a comparative study has been done between the proposed FOA and four other evolutionary algorithms i.e. GA, PSO, Jaya and FPA each running for 100 iterations except for the Forest optimization algorithm, which runs for only 10 iterations.

Table 11 represents the average response time of GA, PSO, Jaya, FPA and FOA using 1000 tasks respectively.

Table 12 represents the total execution time of GA, PSO, Jaya, FPA and FOA using 1000 tasks respectively.

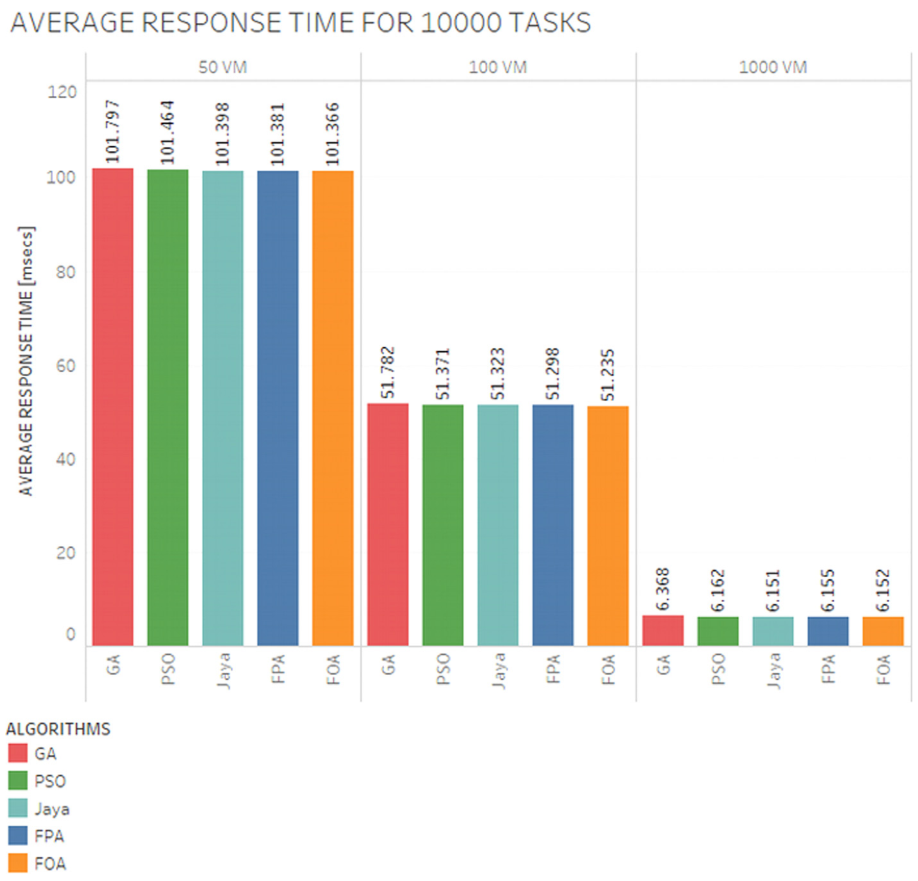

Fig. 8. Average response time (ms) for 10,000 tasks and (a) 50VMs (b) 100VMs (c) 1000 VMs.

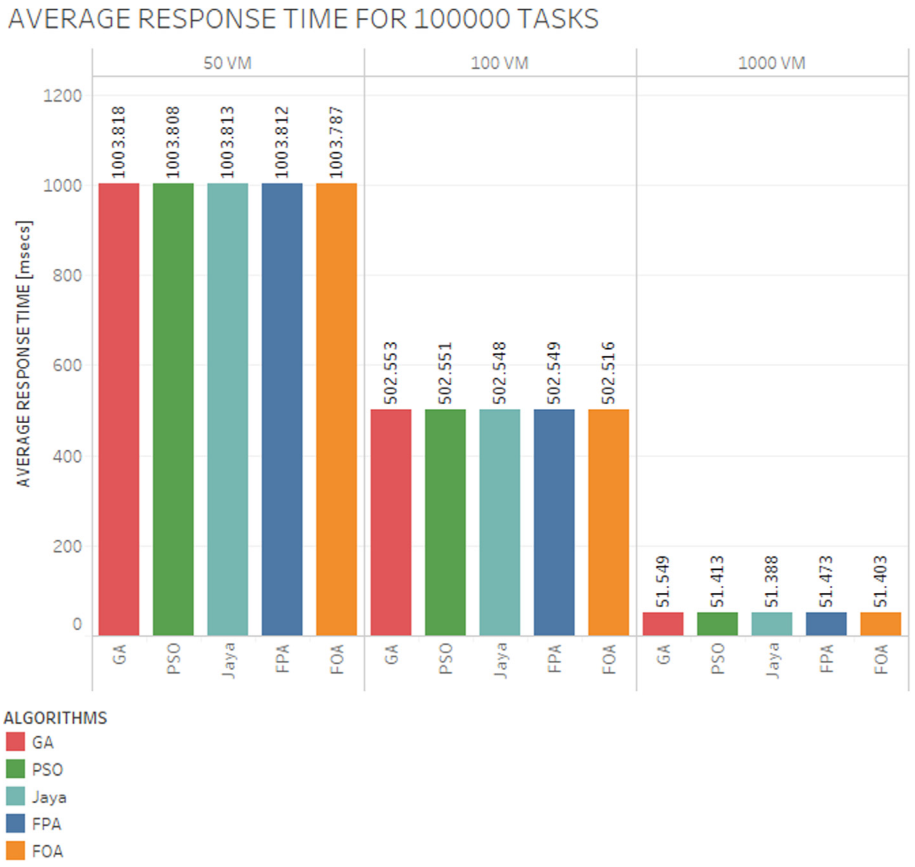

Fig. 9. Average response time (ms) for 100,000 tasks and (a) 50VMs (b) 100VMs (c) 1000 VMs.

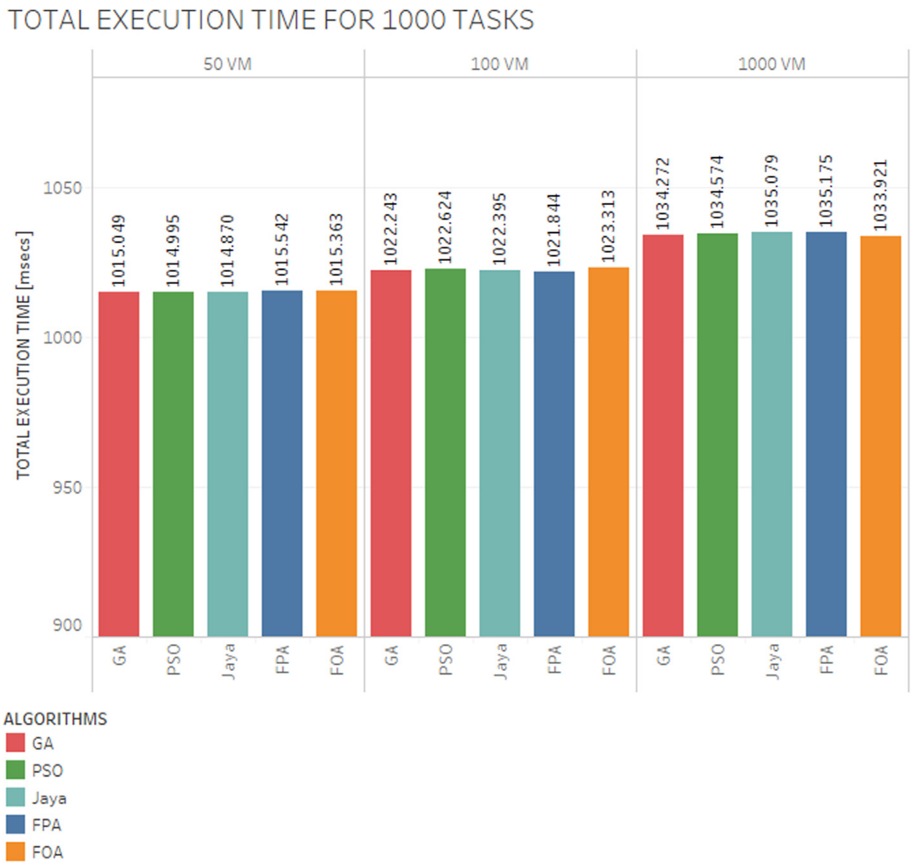

Fig. 10. Total execution time (ms) for 1000 tasks and (a) 50VMs (b) 100VMs (c) 1000 VMs.

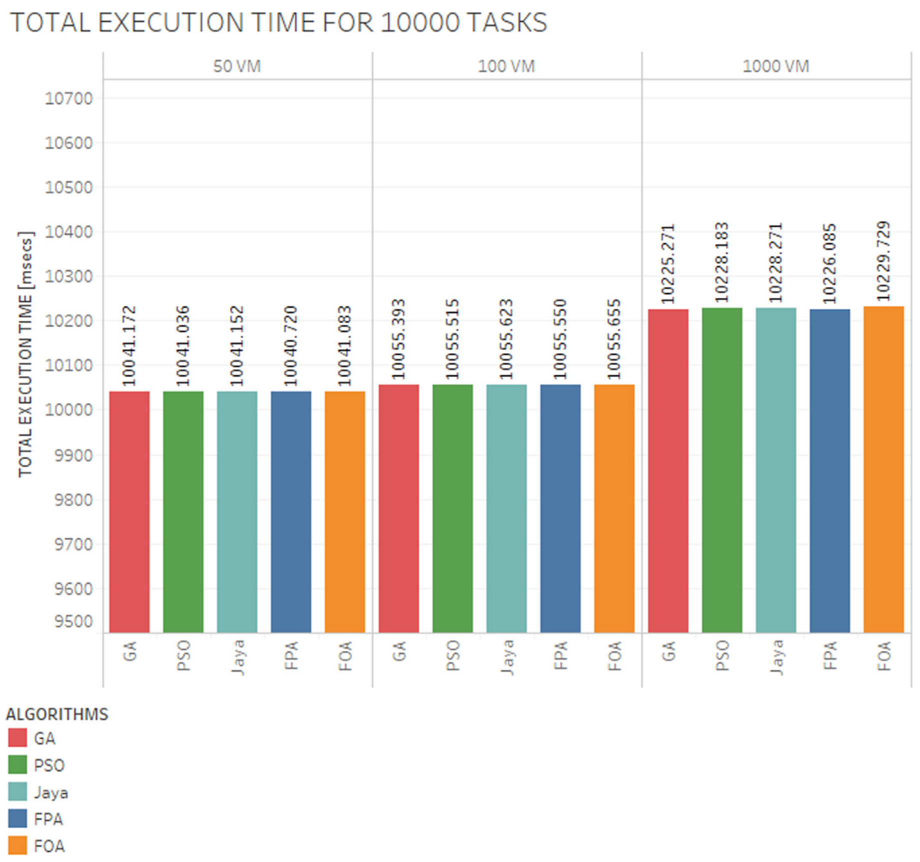

Fig. 11. Total execution time (ms) for 10,000 tasks and (a) 50VMs (b) 100VMs (c) 1000 VMs.

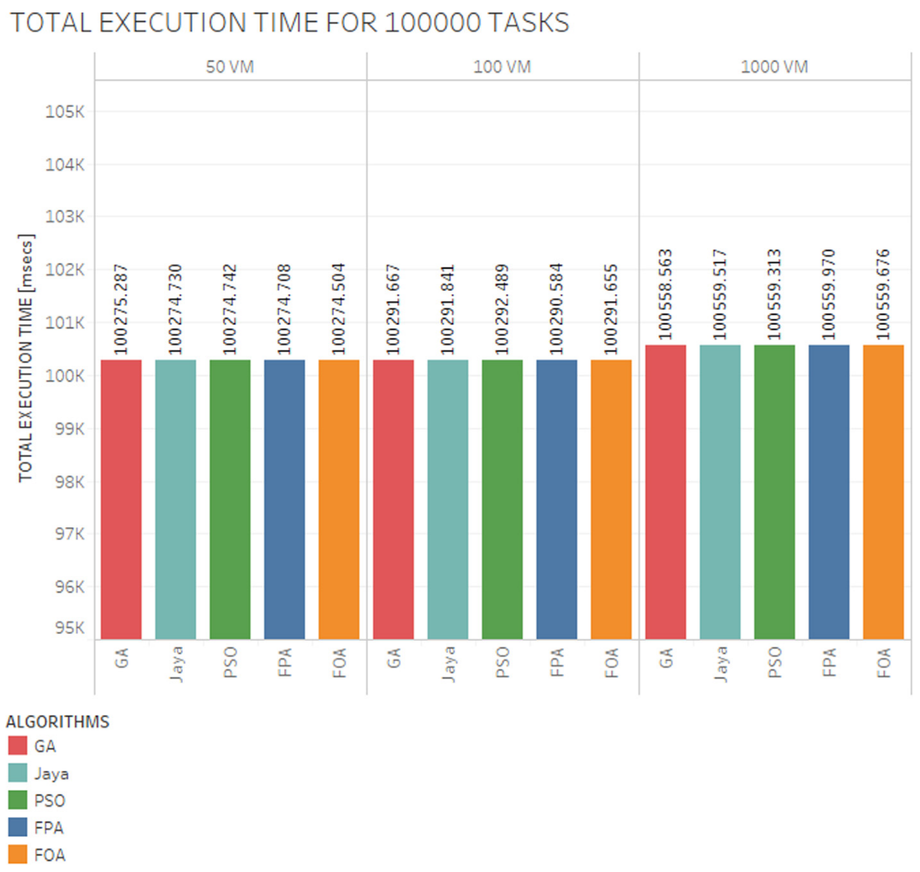

Fig. 12. Total execution time (ms) for 100,000 tasks and (a) 50VMs (b) 100VMs (c) 1000 VMs.

Table 13 represents the average response time of GA, PSO, Jaya, FPA and FOA using 10,000 tasks respectively.

Table 14 represents the total execution time of GA, PSO, Jaya, FPA and FOA using 10,000 tasks respectively.

Table 15 represents the average response time of GA, PSO, Jaya, FPA and FOA using 100,000 tasks respectively.

Table 16 represents the total execution time of GA, PSO, Jaya, FPA and FOA using 100,000 tasks respectively.

From the above tables, it is seen that the proposed FOA performs better than GA, PSO, Jaya and FPA when the tasks to VM ratio is significantly high.

The graphs are drawn considering average response time as shown in Figs. 7, 8 and 9.

The graphs are drawn considering total execution times as shown in Figs. 10, 11 and 12.

7. Conclusion and future work

FOA is enlivened by the function of 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. However most significantly, several seed dispersal techniques help them survive for an extended span of time and expand their territories. A comparison is also done with GA, PSO, Jaya and FPA by conducting experiments to grasp the performance of each of the algorithms. The simulation result shows that the proposed FOA algorithmic program outperforms or matches GA, PSO, JAYA and FPA by minimizing the average response and total execution time under significant load while running for a fraction of their total number of iterations. It provides faster response time, while virtually maintaining the same execution time as that of the others. If the ratio of total number of tasks to that of total number of VMs is considerably high, we see significant performance improvement using the FOA. We shave off notable latency in the response time for each task while sacrificing almost nothing in total execution time. However, as the ratio decreases, the performance degrades. But the parameters can be tweaked to provide a balanced performance under lighter load. So, as a future endeavor, we aim at finding appropriate values of these system parameters for a steady performance under light, moderate and heavy system load.

CRediT authorship contribution statement

Ishan Aryendu: Conceptualization, Data curation, Methodology, Writing - original draft. **Subasish Mohapatra:** Investigation, Visualization. **Subhadarshini Mohanty:** Supervision. **Sachi Nandan Mohanty:** Writing - review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Z. Xiao, W. Song, Q. Chen, Dynamic resource allocation using virtual machines for cloud computing environment, IEEE Transactions on Parallel and Distributed Systems 24 (6) (2013) 1107–1117.

[2] [2] Zenon Chaczko, Venkatesh Mahadevan, Sharzad Aslanzadeh, Christopher Mcdermid (2011) Availability and Load Balancing in Cloud Computing International Conference on Computer and Software Modeling IPCSIT VOL.14 IACSIT PRESS, SINGAPORE 2011

[3] Foster, I., Y. Zhao, I. Raicu and S. Lu, Cloud Computing and Grid Computing 360-degree compared, inproc. GridComputing Environment Workshop.

[4] Buyya R., R. Ranjan, and RN. Calheiros, Intercloud: Utility oriented federation of cloud computing environments for scaling of application services, in Proc. the 10th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP), Busan, South Korea, 2010.

[5] Desai, J. Prajapati, A survey of various load balancing techniques and challenges in cloud computing, Int. J. Sci. Technol. Res. 2 (11) (2013) 158–161.

[6] S.P. Hubbell, Tree dispersion, abundance, and diversity in a tropical dry forest, Sci. 203 (4387) (1979) 1299–1309.

[7] H.F. Howe, S. Judith, Ecology of seed dispersal, Annual Rev. Ecol. Syst.c (1982) 201–228.

[8] D.S. Green, The efficacy of dispersal in relation to safe site density, Oecologia 56 (2–3) (1983) 356–358.

[9] M.L. Cain, B.G. Milligan, A.E. Strand, Long-distance seed dispersal in plant populations, Am. J. Botany 87 (9) (2000) 1217–1227.

[10] S. Ozsen, S. Gunes, Attribute weighting via genetic algorithms for attribute weighted artificial immune system (AWAIS) and its application to heart disease and liver disorders problems, in: Expert Systems with Applications, Elsevier, 2009, pp. 386–392.

[11] John McCall, Genetic algorithms for modelling and optimisation, Journal of Computational and Applied Mathematics, Volume 184, Issue 1, 2005, Pages 205-222, ISSN 0377-0427.

[12] Agoston E. Eiben, James E. Smith, Introduction to evolutionary computing, 53, Springer, Heidelberg, 2003.

[13] Yang, Xin-She & Karamanoglu, Mehmet. (2013). Swarm Intelligence and Bio-Inspired Computation: An Overview. 10.1016/B978-0-12-405163-8.00001-6.

[14] Yang, Xin-She. (2012). Flower Pollination Algorithm (Flower Algorithm). 10.1016/B978-0-12-416743-8.00011-7.

[15] Venkata Rao, Ravipudi, Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems, Int. J. Industr. Eng. Comput. 7 (2016) 19–34, https://doi.org/10.5267/j.ijiec.2015.8.004.

[16] Pandey, Hari. (2016). Jaya a Novel Optimization Algorithm: What, How and Why?. 10.1109/CONFLUENCE.2016.7508215.

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

[18] S. Boyd, L. Vandenberghe, Convex Optimization, Cambridge University Press, Cambridge, 2004.

[19] M. Grotschel, L. Lovasz, Combinatorial optimization (Chapter 28), Handbook of Combinatorics, Elsevier Science, North Holland, 1995.

[20] Manizheh Ghaemi, Feizi Derakhshi, Mohammad Reza, Forest optimization algorithm, Expert Syst. Appl. 41 (2014) 6676–6687, https://doi.org/10.1016/j.eswa.2014.05.009.