Efficient Resource Allocation in Virtualized Cloud Platforms Using Encapsulated Virtualization Based Ant Colony Optimization (EVACO)

Nirmalya Mukhopadhyay, Babul P. Tewari, Dilip Kumar Choubey, and Avijit Bhowmick

Abstract Virtualization in cloud computing ensures maximum utilization of the computational resources by creating virtual instances. In this paper, we have jointly considered virtualization and ant colony optimization (ACO) to propose a novel encapsulated virtualization-based ACO (EVACO) technique that allocates the computational resources efficiently. We have developed an efficient mathematical model that modifies the traditional ACO and encapsulated it in virtualized cloud platform. The objective is to optimize the resource allocation so that execution can be efficient. The novelty of the proposed model has been established through extensive simulations and comparative study. Our algorithm took less execution time in compare to some popular benchmark algorithms. Moreover, the number of iterations and virtual resources required to complete tasks were less than that of other algorithms. It establishes the optimality of our proposed approach.

Keywords EVACO · Virtualization · Cloud computing · Ant colony optimization · Execution model in cloud · Cloud cost model

1 Introduction

Cloud computing has been a twenty-first century marvel that has occupied the entire information technology (IT) world and also many of the non-IT enterprises has shifted their business information to cloud. It has appeared as a virtue to both

N. Mukhopadhyay · B. P. Tewari (✉) · D. K. Choubey
Department of Computer Science & Engineering, Indian Institute of Information Technology, Bhagalpur, India
e-mail: nirmalya.cse.2103004@iiitbh.ac.in; bptewari.cse@iiitbh.ac.in; dkchoubey.cse@iiitbh.ac.in

A. Bhowmick
Department of Computer Science & Engineering, Budge Budge Institute of Technology, Kolkata, India

M. Kumar et al. (eds.), *6G Enabled Fog Computing in IoT*,
https://doi.org/10.1007/978-3-031-30101-8_6

enterprises and academic users [1]. The reasons being obvious that it has helped the business fraternity to reduce their IT overhead. The introduction of this new age technology has revolutionized the concept of computational resource utilization. Hence, a major change in the computational approach has been observed. Different computing devices such as processing units, memory units, storage components, networking segments are virtualized and these virtual instances are used to create cloud infrastructural environment [2]. It has been noticed that the total cost of ownership (TCO) has been in the affordable range for the small and medium size enterprises [3].

Resource optimization in cloud computing has been incorporated in many different ways. Many algorithms have been built and many computational and mathematical models have been designed such as particle swarm optimization, genetic algorithm, Nash equilibrium, cooperative game-based optimization etc. [4–7]. In this context, ant colony optimization (ACO) is also a popular technique for achieving the optimization solution in an efficient manner. It follows probabilistic slant to discover the finest set of solutions from a puddle of solutions. Researchers have developed computational logic to create artificial ants through programming. These artificially programmed agents (resemblance with the behavior and properties of biological social insect ant) ensure a better solution after every iteration through predominant paradigm [8, 9]. Finally, the best solution is obtained and applied for enhancing the overall performance of the system. In our proposed approach, we have investigated resource optimization through a modified ACO technique to optimize the execution cost and time through an efficient resource selection mechanism.

While virtualization will help in mimicking the computational resources into virtual instances [4, 10], and logically isolating and segregating them for keeping individual existence for participating into optimization of resources for cloud computing; the ACO algorithm will be in action to find out the best probable solutions for allocating the resources to the virtual instances. Virtualization will help in maximizing the utilization of the available resources [1, 11] whereas, ACO will allocate the required resources to incoming tasks. So, in this study, we have encapsulated the virtualization technology and ACO algorithm with the objective to reach to an optimized virtual resource allocation method that will efficiently compute all the tasks without affecting the system performance.

Cloud computing is used to store secure end-to-end encrypted IoT data on multiple servers and can be accessed online. To avoid the dependency of internet based centralized cloud storage and edge devices, fog computing creates local networks for decentralized IoT ecosystem to decide whether to process the data locally or remotely. This means, fog computing enhances the opportunity to analyze and process offline data if access to the cloud is not stable or possible. Hence, efficient resource allocation in virtualized cloud platform unfolds a gateway for fog computing to properly place fog nodes into edge platforms that can provide unprecedented processing speed on sensitive and real time operations for data analytics in IoT.

In this study, we have addressed a major issue: whether the resource allocation policy can be optimized in virtual cloud platform or not. So, our objective is to

find out the most feasible resource allocation policy among a pool of solutions that provides best cloud execution model to the cloud users. To ensure this selection, encapsulated virtualization-based ant colony optimization (EVACO) algorithm is proposed for partially virtualized cloud platform. This algorithm ensures the selection of the improved solution in such a way, so that least number of iterations are required to complete the execution. So, in our proposed approach, first the pool of virtualized resources is created as an initial population and then an independent resource will be selected through EVACO algorithm to complete the assigned task with minimum amount of time without affecting the system performance.

The rest of the paper is organized as follows: Section 2, describes the existing works. We have explained the traditional methods in Sect. 3. Our proposed mathematical model and associated EVACO algorithm is formulated in Sect. 4, which is followed by the result analysis in Sect. 5. Finally, we conclude our discussion in Sect. 6.

2 Motivation

Computing has changed its paradigm from time to time. The main objective of this transformation is to efficiently solve real world problems. Furthermore, computing technologies became a part of daily activities. In this regard, computation needs to be intelligent and easily accessible to the users. The variants of distributed system focus on reliable, available and demand-based execution of tasks. Allocation of resource towards effective computation has always been an area of research and development. Different algorithms for different environments and dependencies have been proposed and implemented to enhance the execution process. Virtual cloud platforms make online analysis of IoT data and fog nodes do it offline. In both the platforms, resources are allocated to resolve tasks in competent manner. We started our research with the motivation to make resource allocation optimal for virtualized cloud platforms, so that we get most feasible solutions with unparalleled speed and performance.

3 Related Study

There have been considerable research works focusing on optimizing and analyzing the resource allocation of cloud computing. In such context [12], suggested a task scheduling algorithm based on the ant colony optimization algorithm (ACOA) that efficiently allocates cloud resources to users' jobs within the virtual machines using diversity and reinforcement techniques [13] found that the typical cloud-based strategy was proven to be unable to meet the requirement of low-latency execution of multiple devices in an edge area. Therefore, they planned to incorporate ACOA-based internet of things (IoT) approach that can lead to the number of edge

devices to create an edge data centre. Finally, they proposed a two-level scheduling optimization scheme that will produce an efficient output in timely manner [5] defined a large-scale peer to peer (P2P) grid system that implements an ACOA to identify the location of required resources. The authors of [14] have proposed an efficient IaC-based resource allocation framework for simulated virtual cloud platforms that provides high throughput and effortless cloud resource configuration management.

Mishra et al. [15] demonstrated the re-engineering approaches to migrate the ACOA to modern Intel multi-core architectures to mitigate the factors that has an impact on hardware performance. On similar research [13], presented a new open computing language (OpenCL) based ACOA and [16] have discussed novel parallel algorithms of the well-known ACOA on the multi-core platforms of Intel Xeon Phi co-processor.

The authors of [8] established a strategy for efficient software module clustering in which dependent modules are placed together inside a cluster. According to [2], the computational complexity of ACO if compared to that of genetic algorithm (GA), crops better result as ACO can optimize the services to a higher degree [9] projected an adaptive learning model based on ACO Algorithm to improve stint, price and eminence factors to realize the optimal resource allocation.

On contrary, our research focuses on optimizing the resource allocation towards completion of cloud tasks in a partially virtualized cloud platform efficiently without hampering the SLA and system performance through encapsulating the ACO algorithm with virtualization technology. In this context, we propose a novel EVACO algorithm along with supportive mathematical cloud execution model. Our research shows better performance than that of existing algorithms, which have been proved through extensive simulations.

The following table displays the comprehensive comparison between different researches discussed above (Table 1).

4 Traditional ACO

In order to formulate the proposed EVACO, we have analyzed the basic approach of ACO. There has been a considerable study to incorporate virtualization techniques to increase the infliction of computational resources. Conversely, ACO has also proven to be a great solution towards resource allocation methods.

In ACO algorithm, each ant obtains a starting node that can be considered as it's nest [5, 17, 18]. From this node, the ant selects the next node based on the rules of the algorithm. The ant completes its journey by traversing all the nodes only once and finally returns home at last. The ants can travel the nodes either concurrently or successively. During the journey, each ant places a convinced volume of pheromones on the route. The amount of pheromone to be poured down in the routes depends on the quality of the path selected by the ant; a petite path usually fall-outs with

Table 1 Comprehensive comparison of related studies

Paper	Problem found	Proposed solution	Remarks
[12]	Inefficient resource allocation in cloud platforms	Diversity and reinforcement-based scheduling techniques optimized through ACOA	Provided better result than existing algorithms in conditional cases
[8]	Lack of dependency in cluster-based cloud platforms	Efficient software module clustering strategy	Resource allocation and response time improved
[15]	Reduced hardware efficiency & performance	Migrating ACO to processor-based operations.	Performance of hardware increased
[13]	High latency in execution through edge devices	Creation of edge data center using OpenCL	Latency dropped significantly
[16]	Reduced hardware efficiency & performance	Developed ACOA for multicore Intel Xeon Phi coprocessor	Performance of hardware increased
[5]	Location transparency of cloud resources	Developed large-scale P2P grid system embedded with ACOA for resource location finding	Identification of location becomes efficient
[9]	High price of computing	Proposed adaptive learning model based on ACOA	Improved stint and price
[14]	Reduced performance and throughput because of configuration management overhead during cloud resource provisioning	Proposed an efficient IaC-based resource allocation framework	Improved throughput and performance because of programmed provisioning

more pheromone. The precipitated pheromone undergoes disappearance which is best known as evaporation.

The probability of selecting the succeeding node j by the Ant to arrive at from the current node i is computed as follows [15, 16, 18]:

$$p\left(c_{ij}|s^{p}\right) = \frac{\tau_{ij}^{\alpha *}\ \eta_{ij}^{\beta}}{\sum\limits_{c_{ij} \in N(s^{p})} \tau_{ij}^{\alpha *}\ \eta_{ij}^{\beta}} \tag{1}$$

where, s^p = a partial solution, N is the list of all existing routes from node i to every neighbor node which are yet to be traversed by the ant, c_{ij} is the route from i to j, p is the probability of incidence, t_{ij} is the quantity of pheromone follow on c_{ij}, h_{ij} is calculated as some empirical factor, $\eta_{ij} = Q/d_{ij}$, where d_{ij} is a remoteness factor amid nodes i and j, Q is a constant weight and α and β are the algorithmic constraints.

4.1 Apprising Pheromone Value

The Ants will update the value of the pheromone on the routes connecting the nodes according to the following formula [2, 11, 16, 17]:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{2}$$

where, m is the count of ants, $\Delta\tau^{k} = Q/Lk$ if the ant k traveled the path C^{k}_{ij} between nodes i and j; Q is a persistent weight (it is the set of all neighbor nodes), and L_k is the distance covered by the ant k during travel and is represented as $\Delta\tau^{k} = 0$.

4.2 Evaporation

After finishing the nth trip by the Ants, evaporation will take place in all the available routes between the nodes. This is given by the formula as follows:

$$\tau_{ij}^{n} \leftarrow (1 - \xi)^{*} \tau_{ij}^{n} \tag{3}$$

where, $\xi \subseteq (0,1]$ is the evaporation factor.

5 Proposed Solution

We propose for a modified version of the basic ACO algorithm by encapsulating it in virtualized platform. We call it encapsulating virtualization-based ant colony optimization (EVACO). To implement our algorithm, first, the available computing resources need to be virtualized to create the initial population. These virtualized resources provide maximum resource utilization percentage through isolated instances. Then, we will apply ACO algorithm on the entire pool of isolated yet correlated and fully functional virtual resources. This, in turn will provide us the optimum solution in minimum number of iterations. The entire process will not hamper the overall performance of the cloud platform. To implement our algorithm, we have taken a step-by-step approach. We have described each step in the following discussion:

5.1 Formulating Primary Population for Virtualized Computing Resources

A control variable in the optimization model is the path taken by an ant in any stratum of an N-dimensional space. The N-dimensional array indicates the path taken by the ant. The array can be represented as:

$$VRP = (vr_1, vr_2, vr_3, \ldots, vr_i, \ldots, vr_n) \tag{4}$$

where, virtual resource pool (VRP) is a set of solutions, n is the total number of control variables, and vr_i is the ith control variable in a single VRP. As shown below, each control variable i selects a weight from a discrete domain V_i's predetermined set of weights:

$$V_i = \left(v_{i,1}, v_{i,2}, \ldots, v_{i,d}, \ldots, v_{i,D_i}\right) \tag{5}$$

in which, i = 1,2,3,...,N; V_i is the encoded set of weights for ith control variable, $v_{i,Di}$ = dth conceivable weight for the ith control variable, and D_i = total number of conceivable weights for the ith control variable. The EVACO algorithm begins with generating a MxN matrix at random, where M is the population size of results and N represent the number of control variables. As a result, the pseudo-random solution matrix looks like this:

$$P_{VR} = \begin{bmatrix} VRP_1 \\ VRP_2 \\ \vdots \\ VRP_j \\ \vdots \\ VRP_M \end{bmatrix} = \begin{bmatrix} vr_{1,1} & vr_{1,2} & \ldots & vr_{1,i} & \ldots & vr_{1,N} \\ vr_{2,1} & vr_{2,2} & \ldots & vr_{2,i} & \ldots & vr_{2,N} \\ \vdots & \vdots & \ldots & \vdots & \ldots & \vdots \\ vr_{j,1} & vr_{j,2} & \ldots & vr_{j,i} & \ldots & vr_{j,N} \\ \vdots & \vdots & \ldots & \vdots & \ldots & \vdots \\ vr_{M,1} & vr_{M,2} & \ldots & vr_{M,i} & \ldots & vr_{M,N} \end{bmatrix} \tag{6}$$

in which P_{VR} is the population of virtual resources; VRP_j is the jth solution, $vr_{j,i}$ is the ith control variable of the jth solution, and M is the population size (i.e., the quantity of existing solutions to solve the task). The weight of $vr_{j,i}$ is arbitrarily nominated from a set V_i.

5.2 Apprising Pheromone to the Control Domain

The EVACO looks for information in various parts of the control space and adds it there. New solutions are created erratically depending on the statistical data available in the control domain. The EVACO algorithm assigns pheromone intensity value to each control variable's weight depending on the solution's fitness score.

The higher the pheromone quantity, the more suitable a solution is, and the other way around. In virtualization scenario, if a virtual resource is available and idle, then the best solution is to allocate that resource to solve a problem to increase the utilization of resources in an optimized way. So, the idle index (ϕ) will be used in exchange of pheromone during the solution. The higher idle index is for a virtual resource, the higher the chances of allocating that resource to a pending task.

To apportion pheromone to the discrete domain, N arrays of size $1xD_i$ are used, each array is allotted to one control variable as follows:

$$P_i = \left(p_{i,1}, p_{i,2}, \ldots, p_{i,d}, \ldots, p_{i,D}\right) \tag{7}$$

where, P_i = the pheromone matrix for the ith control variable and $p_{i,d}$ = the pheromone intensity of the dth probable weight of the ith control variable. At the start of the algorithmic refinement, the elements of the matrix Pi equate zero. In virtualization platform, as the idle index is changed for a virtual resource, it becomes either not available or any other higher idle index resource is chosen for next pending task. So, the availability index (σ) is equivalent with evaporation coefficient (ϵ).

In basic ACO algorithm, pheromone intensity for the dth possible weight of the ith control variable is updated as follows [3, 15]:

$$p_{i,d}^{n} = (1-\varepsilon)^{*}\, p_{i,d} + \sum_{j=1}^{M} \Delta c_{i,d}^{j} \tag{8}$$

in which, P^n = the updated intensity of pheromone of the dth possible weight of the control variable, ϵ = the rate of evaporation and $\sum_{j=1}^{M} \Delta c_{i,d}^{j}$ = the quantity of pheromone laid on the dth possible weight of the ith control variable by the jth ant.

The weight of $\sum_{j=1}^{M} \Delta c_{i,d}^{j}$ resembles to the fitness score of the jth solution, and is assessed for our proposed EVACO algorithm as follows in a minimization problem:

$$\Delta c_{i,d}^{j} = \begin{cases} \frac{Q}{F(VRP_j)} & if\ vr_{j,i} = v_{i,d} \\ 0 & if\ otherwise \end{cases} \tag{9}$$

where $j = 1, 2, \ldots, M$; $i = 1, 2, \ldots, N$; $d = 1, 2, \ldots, D_i$; Q is a constant weight and $F(VRP_j)$ is the fitness weight of the jth solution.

In the EVACO algorithm, we can replace the pheromone weight $p_{i,d}$ with idle index (ϕ) and the evaporation coefficient (ϵ) with availability index (σ). So, the modified equation can be derived from Eqs. (8) and (9) as follows:

$$\text{maximize}\,(\phi) = \left\lfloor (1-\sigma)^{*}\,\phi + \sum_{j=1}^{M} \frac{Q}{F\left(VRP_j\right)} \right\rfloor \tag{10}$$

This equation will consider a lower edge weight which is bare minimum for idle index as we have used a floor function to set the lower limit. Any weight below a predefined constant will be disregard and hence the performance degradation cannot occur. If and only if the idle index of a computational resource is higher than the lower bound, then EVACO will consider that resource for further update.

5.3 Generation of Updated Solution

There is always an infinitesimal probability of never nourishing the condition p (c_{ij} | s^p) nj, which can be deduced from equation number 1. So, in this research, we have used some amendment in the traditional ACO Algorithm. For the given node i, we create a random number $n_j \subset [0,1)$ and then compare that to the moving sum. The result is updated whenever is recalculated. When $n_j p$ (c_{ij} | s^p) satisfies, considered as the index of the next node j to travel. Essentially, this difference is only noticeable in the beginning phases of the test, when the amount of pheromone trailing on the various pathways is fairly similar. The summation of the probabilities of the conceivable weights of each control variable is equal to one. So, we write:

$$\sum_{d=1}^{D_i} \Psi_{i,d} = 1 \tag{11}$$

The weights of the control variables of an updated result are arbitrarily elected depending on the assessed likelihood. To do so, we calculated a cumulative probability for each potential weight of each judgement variable, as shown below:

$$\Upsilon_{i,r} = \sum_{d=1}^{r} \Psi_{i,d} \tag{12}$$

in which, $\Upsilon_{i,r}$ is the accretive likelihood of the rth possible weight of the ith control variable. Then, an arbitrary weight (ω) is inferred within the range of [0,1]. Depending on the comparative values obtained, weights are selected. For example, if the weight of ω is less than the weight of $\Upsilon_{i,r}$, $\Upsilon_{i,1}$ is selected; otherwise, the rth weight is selected in a pattern where $\Upsilon_{i,r} - 1 < \omega < \Upsilon_{i,r}$.

5.4 Mutating EVACO to Optimize Further

The worst possible situation that may occur in any optimization algorithm is when it gets trapped in some local optimal loop. Hence, the notion of mutation is brought from the genetic algorithm (GA) to avoid being locked into the local minima. When an ant has finished its trip, it will begin the mutation process based on the mutation probability. A node is randomly eliminated from the tour and is replaced by another

Table 2 Symbols used in EVACO algorithm and their descriptions

Symbols	Description
M	Initial population size (set of all virtual instances)
P_{VR}	Randomly generated population of virtual instances
n	# control variables
V_i	# possible weights for control variable i
TQ	Task queue (used as source node)
R_v	Virtual resource (used as destination node)
VRP	Decomposed tasks (used as ants)
AL	Availability list

randomly picked node from the same group. Finally, the randomly chosen node is added into the $(m\ 1)$ positions. The new solution of the afflicted ant is the shortest tour of all the feasible tours, including the initial route. As a result, the total number of iterations reduces up to a great extent resulting optimal solution. The outline of the proposed EVACO is represented in Algorithm 1. Note that, Table 2 represents different notations used in this algorithm.

Algorithm 1: Proposed EVACO

```
Input: M, P_VR, N, V_i, TQ, R_v, VRP, AL
 Output: Optimal resource allocation to the decomposed task
 Initialize M=N=AL=0
 begin
 generate M initial population P_VR of possible solutions randomly
while termination criteria stand false do {
 evaluate fitness values for all the existing solutions
 assign idle index to decision variables based on fitness values
 finalize availability index to select if a resource will enter
AL or not
 allocate idle resources to VRP
 update AL by removing the already allocated resources
for i: = 1 to N do {
 for d: =1 to V_i do {
 update idle index of possible value d_ij for decision variable i
                      based on availability index and fitness value
 evaluate selection probability of possible value d_ij
 }
 }
for k: = 1 to M do {
 for i: = 1 to N do {
 select a random value r_ki for i^th decision variable among all
possible values based on their probabilities
 }
 }
 }
select a solution if the fitness value is higher than initial
solution
 update P_VR
 mutate to exit the local minima for optimize further
 }
```

5.5 Flowchart and Service Model

A flowchart is a pictorial representation of flow of control. For our proposed EVACO algorithm, we have presented a flowchart that make the concept easier for implementation. Figure 1 below, is the technical flowchart for our algorithm.

We, moreover, have designed a service model for our proposal that shows how the subtasks are assigned with virtual resources efficiently using EVACO algorithm to produce optimal solution for the cloud customer. Figure 2 shows the service model for our proposal.

6 Result Analysis

In this section, we evaluate our proposed EVACO algorithm to prove its expediency. In order to evaluate the performance of the proposed approach, we have considered a methodological approach to create an environment for our simulations. We have developed a customized java program to implement the proposed EVACO algorithm. We have first discussed about the simulation set-up followed by the analysis of the results obtained from the simulations. Finally, the proposed EVACO algorithm is compared with some existing approaches from the literature.

6.1 Simulation Set-Up

Our experiment for evaluating the proposed EVACO algorithm and the proposed execution model requires both hardware and software set-ups. Using the hard- ware set-up, we have created a virtualized cloud platform. The detail of hardware set-up is listed in Table 3.

The software module for our simulation is installed using the hardware specifications mentioned in Table 4. As we have to apply our proposed algorithm in a virtualized platform, we have used VMWare workstation as type-II hypervisor. We have installed it on the top of Windows 10 operating system. After installing the workstation, we have created host virtual machines using ESXi server virtualization framework.

To complete the simulation, we have started with creating the virtual instances inside the physical computer system. We have used the set-up described in Table 3 for this step. We have implemented EVACO algorithm through a customized java code, running in our VMs. The primary intention is to find a set of optimum resources that can be allocated to each and every decomposed tasks. While implementing the code, we have taken care of all the constraints of EVACO algorithm. Our proposed algorithm is a direct enactment of our proposed mathematical model.

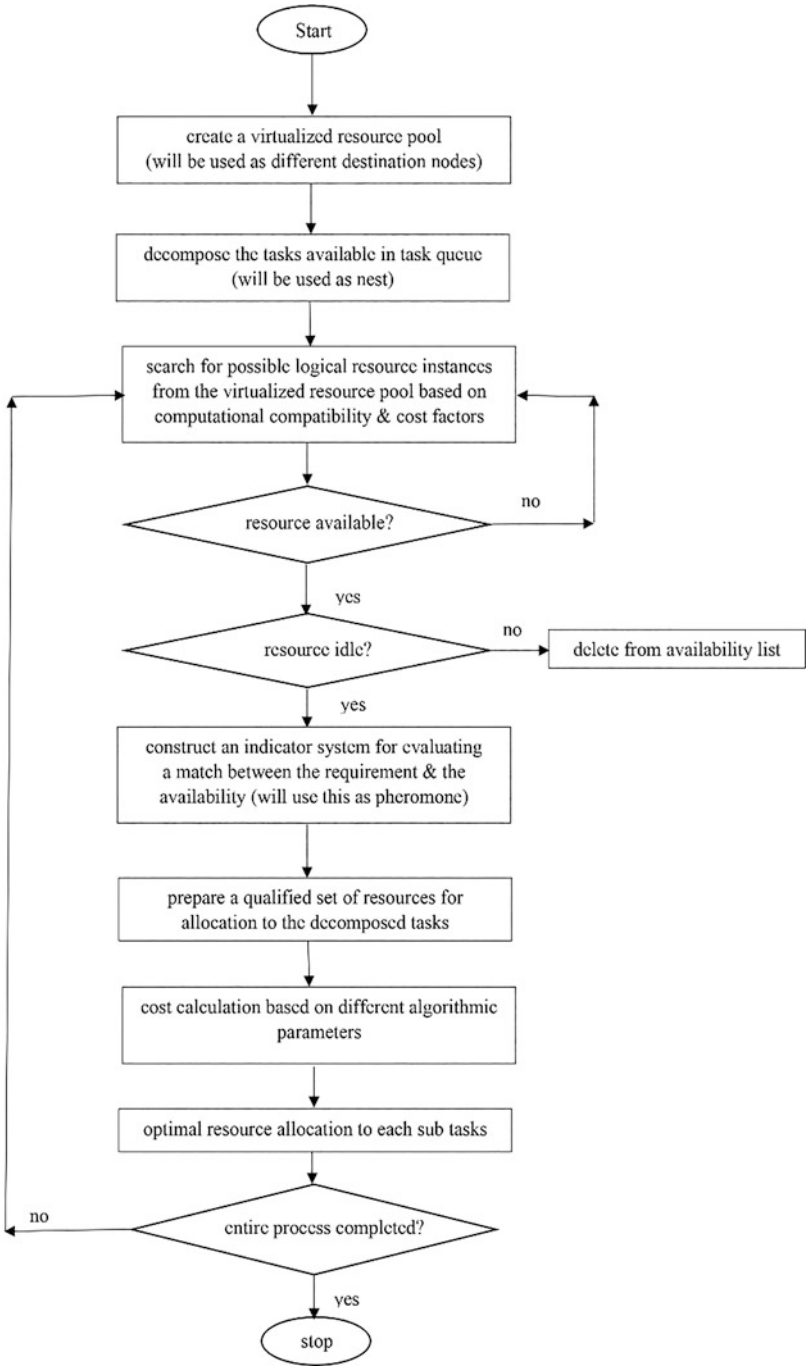

Fig. 1 Flowchart for EVACO algorithm

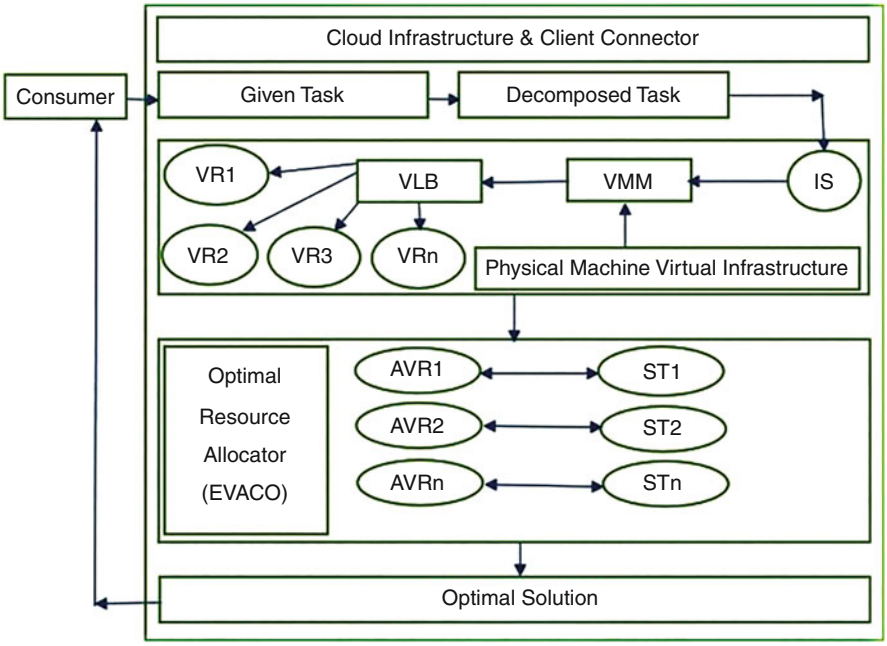

Fig. 2 Proposed service model

So, while writing the java code, we have taken care of all the required parameters for execution.

6.2 Simulation Result

Performance Evaluation of EVACO Algorithm
In order to evaluate the performance of our proposed EVACO algorithm, we focus on the parameter of resource allocation to a sub task and minimization of time to complete the entire process. The algorithm is designed to reach to the solution with minimum number of iterations. Additionally, through mutations it avoids of getting trapped inside the local minima.

We have started our experiment with 20 virtual resources and arbitrarily 186 number of subtasks. These virtual resources have been created inside a host. Our aim in the algorithm is that whenever a subtask will appear in the queue, it will be allocated to the resource that is most feasible. So, the virtual resources (also called nodes) are placed randomly in a predefined partially virtualized cloud platform.

The starting node, marked as 1 in Fig. 3a and Fig. 3b also chosen randomly and represents the comparative node traversal routes in two different iterations. From

Table 3 Details of hardware set-up for simulation

Parameters	Weight/range	Specification
# data centers	5	Simulated within the physical machine
# hosts	5	Running inside the host
RAM	16,384 MB	Double data rate V4
Host RAM	12,288 MB	Double data rate V4
GFX card	4096 MB	Provides additional processing power
Host N/W bandwidth	500–2000 MB/s	Network card of physical machine is used
# CPUs	[2, 8]	Intel i7 octa core 3.06 GHz
# VMs	40	Used VMWare workstation, ESXi v6.7
# vCPUs	[1, 4]	Assigned virtually to VM using VMM
Capacity of vCPU	[1000, 2500] MIPS	Assigned virtually to VM using VMM
Capacity of vRAM	[512, 4096] MB	Assigned virtually to VM using VMM
Capacity of VM bandwidth	[500, 2000] MB/s	Assigned virtually to VM using VMM
Direct attached storage	1,048,576 MB	SATA HDD
Virtual hard disk for host	358,400 MB	Virtual hard disk created & managed
VHD for VM	102,400 MB	Virtual hard disk assigned to VM
# cloud tasks	[100, 700]	Taken through ispell dataset ftp://gnu.mirror.iweb.com/
MIPS of vCPU	[100, 20,000]	vCPU speed measured in MIPS
Size of task files	[200, 400] MB	Located inside VHD
Size of output file	[20, 40] MB	Saved inside VHD

Table 4 Details of software set-up specifications

Parameters	Specification/description
File system	Standard virtual machine file system
Processor	Intel VT-x 64-bit x86 NX/XD bit enabled processor with at least 2 cores
Main memory	8 GB vRAM
Ethernet controller	1 GB
VHD	SCSI 350 GB
Boot partition	32 GB HDD with 8 GB USB drives embedded
VMware tools	ESX-OS data volume tool is migrated from locker partition, used for core dump volume and finally the partition is rubbed out

Fig. 3a to Fig. 3b, it can be noticed that for different iterations the starting nodes have changed to get the optimized shortest traversal route for the given parameters. For example, the starting node (marked as 1) in Fig. 3a has become node number 5 in Fig. 3b and the node number 16 in Fig. 3a has become the starting node in Fig. 3b. This happened because we have given different parameter list for EVACO algorithm for execution in these two cases. We further observe that the choice of next node changes with the change in starting node. The traversing of nodes from 1 to 20 is an implication of the fact that these virtual instances (called as nodes) are available to be allocated to the incoming sub tasks. This is possible because the idle

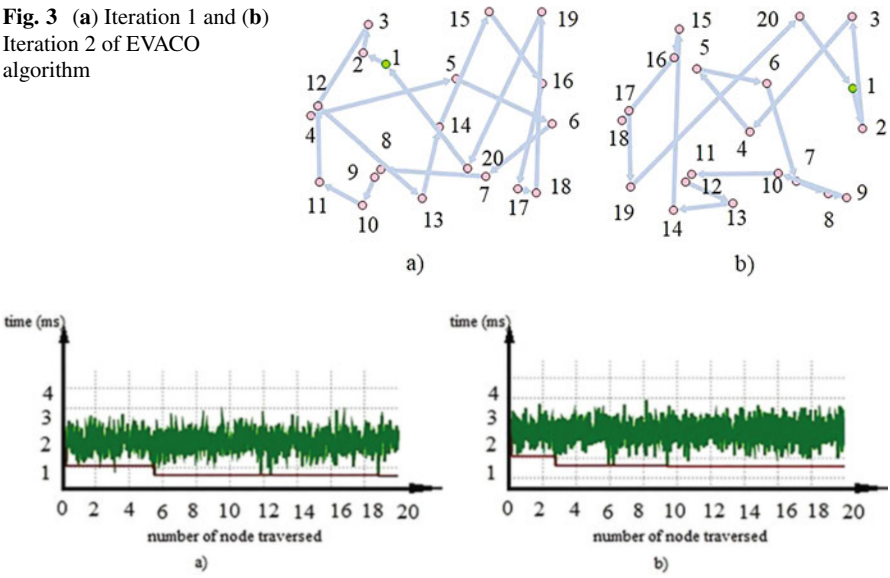

Fig. 3 (**a**) Iteration 1 and (**b**) Iteration 2 of EVACO algorithm

Fig. 4 Performance comparison graph of (**a**) Iteration 1 and (**b**) Iteration 2 of EVACO algorithm

index (as described in the Eq. 10) of these nodes are high and hence sub tasks are allocated to them.

The performance comparison for the stated iterations is demonstrated in Fig. 4a and Fig. 4b which reflects the time (in milliseconds) required to traverse all 20 nodes. The numeric weights of the time requirement to execute the proposed algorithm have been shown in Table 4 for further comparisons.

From Table 4, it is clearly noticeable that EVACO algorithm substantially provides better time complexity than that of other algorithms like PSO, GA, IACO, SACO, ACO etc. We have calculated mean, variance and standard deviation weights from the obtained result to compare these algorithms in terms of computational time spent for different iterations. The detailed graphical comparisons have been displayed in Figs. 5, 6, 7, 8, 9 and 10. From the generated graphs, we can claim that our proposed EVACO algorithm and the associated mathematical model stands higher than the existing approaches (Table 5).

7 Conclusion

ACO has been able to show how effectively resources can be optimized for improved allocation policy. On the other hand, virtualization proved its significance to cloud computing by reducing the infrastructure overhead and increasing the throughput of the data centres. Combined, they can be used for enhancing the

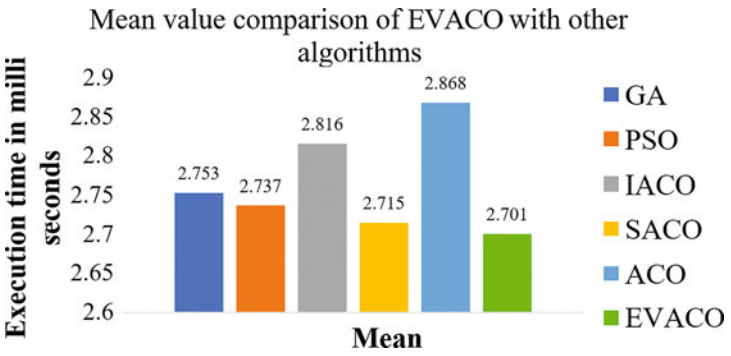

Fig. 5 Comparison of mean for PSO, GA, IACO, SACO, ACO and EVACO

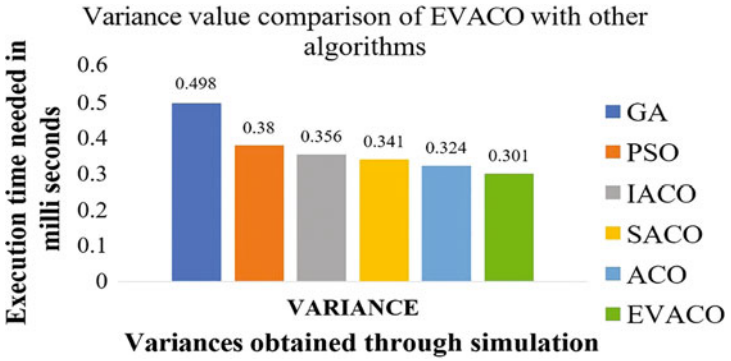

Fig. 6 Comparison of variance for PSO, GA, IACO, SACO, ACO and EVACO

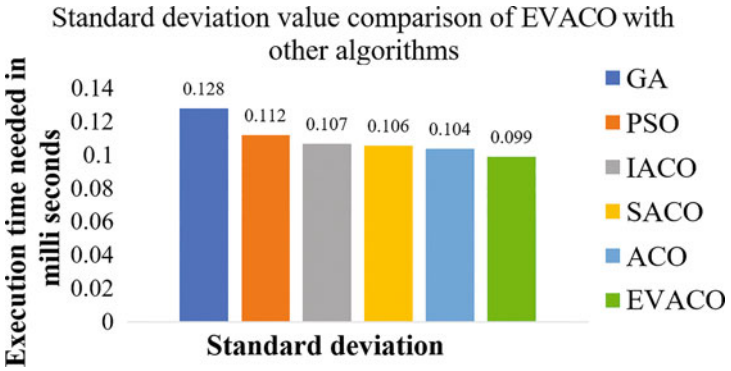

Fig. 7 Comparison of standard deviation for PSO, GA, IACO, SACO, ACO and EVACO

resource optimization policy in partially virtualized cloud platforms. In this paper, we have proposed a novel algorithm (called as EVACO) that applies ACO in a modified approach in a partially virtualized cloud platform for finding feasible

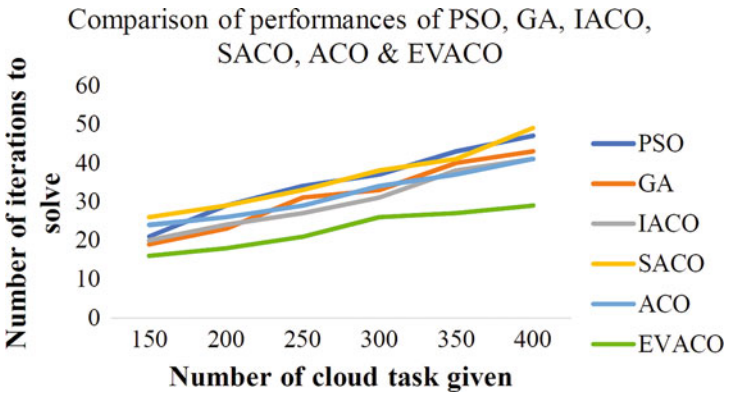

Fig. 8 Comparison of performance analysis for PSO, GA, IACO, SACO, ACO and EVACO

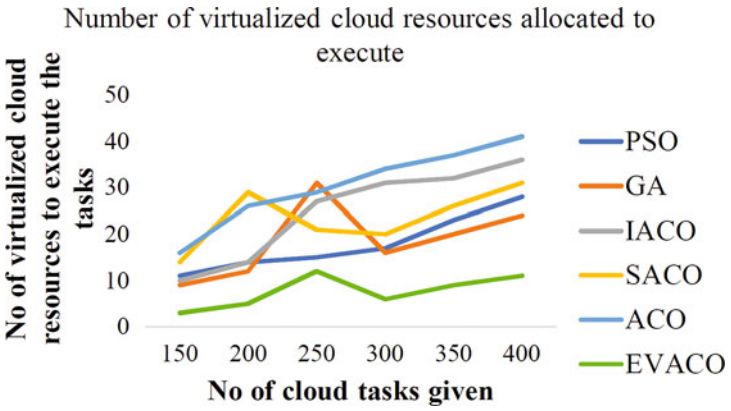

Fig. 9 Comparison of resource requirements for PSO, GA, IACO, SACO, ACO and EVACO

Fig. 10 Comparison of makespan requirements for PSO, GA, IACO, SACO, ACO and EVACO

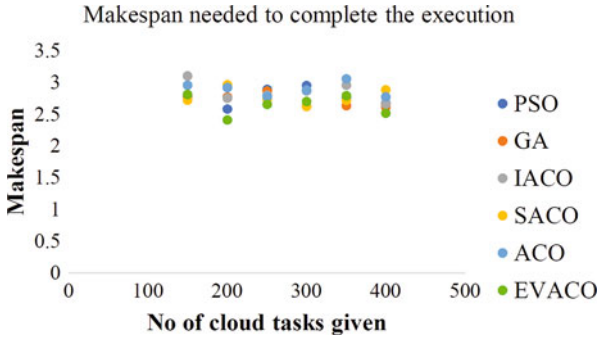

Table 5 Time comparison of EVACO vs. other algorithms for 30 iterations

#Iterations	PSO	GA	IACO	SACO	ACO	EVACO
1	2.753	2.871	2.748	2.846	2.949	2.617
2	2.862	2.510	2.800	2.943	2.730	2.503
3	2.958	2.677	2.999	2.981	3.079	2.714
4	2.660	2.675	2.823	2.861	2.926	2.666
5	2.801	2.779	2.722	2.650	2.909	2.709
6	2.603	2.619	2.791	2.797	2.814	2.699
7	2.733	2.721	2.591	2.955	2.804	2.805
8	2.688	2.882	2.820	2.917	2.744	2.635
9	2.683	2.871	2.720	2.519	2.904	2.800
10	2.888	2.685	2.963	2.520	2.906	2.741
11	2.619	2.742	2.729	2.785	2.792	2.673
12	2.748	2.619	2.519	2.578	2.720	2.720
13	2.800	2.709	2.677	2.889	2.817	2.811
14	2.999	2.835	2.675	2.952	3.030	2.943
15	2.823	3.101	2.779	2.760	2.858	2.814
16	2.722	2.752	2.619	2.669	2.961	2.638
17	2.791	2.795	2.721	2.617	2.728	2.521
18	2.591	2.714	2.882	2.503	2.940	2.606
19	2.820	2.605	2.871	2.714	2.846	2.713
20	2.720	2.706	2.685	2.823	2.943	2.507
21	2.963	2.828	2.742	2.722	2.981	2.825
22	2.729	2.707	2.619	2.791	2.861	2.639
23	2.519	2.693	2.709	2.591	2.650	2.670
24	2.520	2.754	2.835	2.820	2.797	2.598
25	2.785	2.774	3.101	2.720	2.955	2.813
26	2.578	2.768	2.752	2.963	2.917	2.409
27	2.889	2.860	2.795	2.729	2.775	2.654
28	2.952	2.629	2.862	2.619	2.879	2.701
29	2.760	2.633	2.958	2.721	3.053	2.789
30	2.669	2.611	2.660	2.882	2.769	2.513

resource allocation policy. In this context, we have also designed a new utilitarian mathematical model. This model shows how a cloud user can get the optimized solution for a given set of tasks. We have run rigorous simulations by setting up the environment using cloud analyst simulator. The simulation results shown in Table 4 explains that the performance of our proposed algorithm is quite satisfactory. We have compared our simulation results with other existing algorithms and found that our proposed algorithm is giving better results than others. Overall, through the simulation we have shown that the performance of our proposed EVACO algorithm is excellent and is better than other existing algorithms in terms of optimal resource allocation and overall execution model.

References

1. Shyam, G. K., & Chandrakar, I. (2018). Resource allocation in cloud computing using optimization techniques. In B. Mishra, H. Das, S. Dehuri, & A. Jagadev (Eds.), *Cloud computing for optimization: Foundations, applications, and challenges. Studies in Big Data* (p. 39). Springer. 10.1007/978-3-319-73676-12.
2. Mishra, A. K., Umrao, B. K., & Yadav, D. K. (2018). A survey on optimal utilization of preemptible VM instances in cloud computing. *The Journal of Supercomputing, 74*, 5980–6032. https://doi.org/10.1007/s11227-018-2509-0
3. Choudhary, A., Gupta, I., Singh, V., & Jana, P. K. (2018). A GSA based hybrid algorithm for bi-objective workflow scheduling in cloud computing. *Future Generation Computer Systems*. https://doi.org/10.1016/j.future.2018.01.005
4. Guerrero, G. D., Cecilia, J. M., Llanes, A., Garca, J. M., Amos, M., & Ujald, M. (2014). Comparative evaluation of platforms for parallel Ant Colony Optimization. *The Journal of Supercomputing, 69*(1), 318–329. https://doi.org/10.1007/s11227-014-1154-5
5. Asgari, S., Jamali, S., Fotohi, R., et al. (2021). Performance-aware placement and chaining scheme for virtualized network functions: A particle swarm optimization approach. *The Journal of Supercomputing*. https://doi.org/10.1007/s11227-021-03758-9
6. Yu, J. F., Li, Y., Yu, H. S., & Shen, Q. (2019). Resources optimization deployment in collaborative manufacturing project based on adaptive Ant Colony Algorithm computer integrated manufacturing systems*, 14*, 3.
7. Munir, A., Koushanfar, F., Gordon-Ross, A., et al. (2013). High-performance optimizations on tiled many-core embedded systems: A matrix multiplication case study. *The Journal of Supercomputing, 66*, 431–487. https://doi.org/10.1007/s11227-013-0916-9
8. Li, C., Wang, C., & Luo, Y. (2020). An efficient scheduling optimization strategy for improving consistency maintenance in edge cloud environment. *The Journal of Supercomputing*. https://doi.org/10.1007/s11227-019-03133-9
9. Takouna, I., Sachs, K., & Meinel, C. (2014). Multiperiod robust optimization for proactive resource provisioning in virtualized data centers. *The Journal of Supercomputing, 70*, 1514–1536. https://doi.org/10.1007/s11227-014-1246-2
10. Tirado, F., Barrientos, R. J., Gonzalez, P., & Mora, M. (2017). Efficient exploitation of the Xeon Phi architecture for the Ant Colony Optimization (ACO) metaheuristic. *The Journal of Supercomputing, 73*(11), 5053–5070. https://doi.org/10.1007/s11227-017-2124-5
11. Farshin, A., & Sharifian, S. (2019). A modified knowledge-based ant colony algorithm for virtual machine placement and simultaneous routing of NFV in distributed cloud architecture. *The Journal of Supercomputing*. https://doi.org/10.1007/s11227-019-02804-x
12. Moon, Y., Yu, H., Gil, J. M., et al. (2017). A slave ants-based ant colony optimization algorithm for task scheduling in cloud computing environments. *Human-centric Computing and Information Sciences, 7*, 28. https://doi.org/10.1186/s13673-017-0109-2
13. Wang, J., Cao, J., Sherratt, R. S., & Park, J. H. (2017). An improved ant colony optimization-based approach with mobile sink for wireless sensor networks. *The Journal of Supercomputing*. https://doi.org/10.1007/s11227-017-2115-6
14. Mukhopadhyay, N., & Tewari, B. P. (2022). Efficient IaC-based resource allocation for virtualized cloud platforms. In I. Woungang, S. K. Dhurandher, K. K. Pattanaik, A. Verma, & P. Verma (Eds.), *Advanced network technologies and intelligent computing. ANTIC 2021. Communications in computer and information science* (Vol. 1534). Springer. https://doi.org/10.1007/978-3-030-96040-7_16
15. Mishra, S., Sahoo, M. N., Sangaiah, A. K., & Bakshi, S. (2019). Nature-inspired cost optimization for enterprise cloud systems using joint allocation of resources. *Enterprise Information Systems, 1-23*. https://doi.org/10.1080/17517575.2019.1605001
16. Nguyen, T. A., Min, D., & Choi, E. (2018). A comprehensive evaluation of availability and operational cost for a virtualized server system using stochastic reward nets. *The Journal of Supercomputing, 74*, 222–276. https://doi.org/10.1007/s11227-017-2127-2

17. Dorigo, M., & Blum, C. (2005). Ant colony optimization theory: A survey. *Theoretical Computer Science, 344*(23), 243–278.
18. Li, C., Wang, C., & Luo, Y. (2020). An efficient scheduling optimization strategy for improving consistency maintenance in edge cloud environment. *The Journal of Supercomputing, 76*, 6941–6968. https://doi.org/10.1007/s11227-019-03133-9