Chapter 4
Energy-Efficient Resource Management of Virtual Machine in Cloud Infrastructure

H. Priyanka and Mary Cherian

Contents

4.1 Introduction

Cloud computing has evolved as a computational tool for large enterprises because of the advancement in the Internet and virtualization technologies. Cloud computing can be defined "as a type of distributed system which consists of a collection of dynamically sourced resources with interconnected and virtualized computers" [1]. There is a huge demand for the cloud services in large industries. To meet the customer requirements, huge number of servers are needed at the data center. This consumes more energy and emission of carbon dioxide also increases. Cloud computing possesses some obstacles such as reliability, management of resources, security, efficiency, and power consumption [2]. One of the resource management

H. Priyanka (✉)
CSE R&D Centre, Dr. Ambedkar Institute of Technology, Bengaluru, Karnataka, India

M. Cherian
Department of CSE, Dr. Ambedkar Institute of Technology, Bengaluru, Karnataka, India

R. Buyya et al. (eds.), *New Frontiers in Cloud Computing and Internet of Things*, Internet of Things, https://doi.org/10.1007/978-3-031-05528-7_4

challenges is scheduling of tasks. Task scheduling refers to allocating cloudlets (tasks) to the resources which are accessible, this will improve the performance and maximize resource utilization [3, 4].

We have proposed a conceptual framework to reduce power consumption and improve the utilization of infrastructural resources of the data center. The framework efficiently assists in provisioning the resources and minimizing the migrations of a virtual instance.

The framework is designed with the help of a hybrid data center with highly configured infrastructural resources [5] for the experimental evaluation of our proposed algorithms. It monitors the processes of cloud environments dynamically, collects, and updates all the characteristics of data center frequently. In the proposed framework, the task scheduling initially receives the workloads with the help of the user's demand. It considers the configuration of Virtual Machines (VMs) and tasks by using the Expected Computation Time (ECT) matrix to generate an initial population.

The Genetically Enhanced and Shuffled Frog Leaping Algorithm (GESFLA) is proposed to select the optimal VMs to schedule the tasks and allocate them into Physical Machines (PMs). The proposed data center monitoring system checks the host utilization and observes the state and mode of it. If a host is in high or low loaded state by over or under-utilization of tasks in an active mode, the VMs are migrated from these hosts to imbalanced or idle hosts. The proposed algorithm executes applications through efficient resource management. Also, the proposed GESFLA reduces power consumption, improves the utilization of cloud infrastructures with minimum migration of VMs. This proposed algorithm performs better than the Genetic Algorithm and Particle Swarm Optimization (GAPSO) algorithm [6].

The virtualization of computer systems and use of servers are increased, thereby leading to increased power consumption and bandwidth demand in the data centers. For the migration of the VMs, transfer time also increases. This raises the data center's costs, carbon dioxide emissions and leads to global warming.

To address these problems, Genetic Algorithm (GA) [7] and SFLA [8] are chosen because GA generates uniform initial population and it can be applied for optimization problems. It is helpful as it gives a better solution. The GA cannot be applied for large optimization problems as it selects random crossover point for transferring of VMs and convergence time is also more. In our proposed method, we have combined SFLA with GA because of following advantages of SFLA:

(i) The searching capability is faster.
(ii) It is robust because it does the shuffling of frog memplexes and results in good solution.
(iii) It has early convergence criteria.

The GA and SFLA are combined to get better results. The genetic algorithm convergence is slow, which can be overcome by applying SFLA. The GESFLA has following advantages in comparison with GAPSO.

(i) Efficient optimization of task scheduling is carried out on the server.
(ii) GAPSO algorithm takes relatively large computation time.
(iii) As it takes more time to terminate, VM migration time is also increased and the resources are not utilized efficiently.

The research objective is stated below. The various phases of GESFLA are explained in Sect. 4.4.

The goals and objectives of research are as follows:

(i) The applications running inside the virtual machine are tracked, and the load should be stored uniformly between all the virtual machines.
(ii) To reduce the execution time of the task and transfer time of the VM, by tracking the usage of cloud resources of applications (tasks) running within VMs.
(iii) To reduce Cloud data center power consumption by efficient scheduling of resources, VM migration allows cloud operators to achieve various resource management goals, such as green computing, load balancing, and maintenance of real-time servers.

This chapter is organized as follows: Section 4.2 elaborates the background of existing works. Section 4.3 explains the proposed methodology GESFLA and its different phases and algorithms. Section 4.4 evaluates the experimental results. Finally, in Sect. 4.5, this chapter is concluded and also future work is discussed.

4.2 Background

Multi-server energy consumption strategy aims at reducing the power usage of the data center by using the least amount of resources that are available in the PMs while switching off inactive PMs at the data center. Man et al. [9] and Ajiro et al. [10] used different methods, such as "First Fit," "Best Fit," to solve VMs allocation problem. A global optimum solution is obtained from the heuristic approaches. "Best fit heuristic solution" is inaccessible because it takes more convergence time, and the proposed work has one more limitation based on a "single objective function." Beloglazov et al. [11] recommended a "Modified Best Fit Decreasing (MBFD) algorithm" by arranging the ascending order of PMs and descending order of VMs based on their processing capability. In the "First Fit Decreasing (FFD)," VM allocation on PMs is done after VMs and PMs are sorted. The drawbacks are: Distinct allocation goal for VMs and MBFD is not flexible as huge requested VMs get influenced at the data center. The VM allocation issue is solved by using different types of algorithms. Many researchers employed bio-inspired and evolution-inspired algorithms for the cloud data center for allocation of VMs, such as "Genetic Algorithm (GA) [12]," "Particle swarm optimization (PSO)," etc. Xiong et al. [13] used PSO to resolve the issue of allocating energy-efficient VMs. The authors considered a single VM type as it is the major drawback of this research.

"Ant colony"-based VM is proposed for VM allocation by the cloud data centers by Gao et al. [14]. The authors used only a single VM and PM combination which is the drawback of their work. Wang et al. [15] recommended the use of PSO to locate efficient VMs in a data center. The drawback of this research is that the allocations of VMs change the particle velocity that requires more iterations and gives an inaccurate result. "Particle Swarm Optimization (PSO) algorithm" [16] is derived from bird flocking, its a bio-inspired algorithm [17]. It is through PSO that each particle in the swarm gives a solution with four dimensions, its present location, the position, its speed, and the location found among all the particles in the population. Its location is set in the search area based on the position its neighborhood has reached. The limitations of PSO algorithms suffer from the partial optimism that triggers less accurate velocity and distance control. PSO is unable to address particle issues in the field of energy consumption. Sharma et al. proposed "Genetic algorithm and Particle Swarm Optimization (GAPSO)" [6, 8], hybridization of GA and PSO results in increasing the fitness value of the parent chromosomes, and thus allocation of VMs to the PMs is achieved in lesser time.The limitation of GAPSO algorithm takes more convergence time.

4.3 Proposed Work

In this section, the proposed method is discussed with different algorithms. Figure 4.1 shows the different phases of GESFLA. To efficiently carry out the process of migration, VM allocation and VM placement, the GESFLA is designed with various phases as follows:

1. Design of Cloud environment
2. Selection of workload
3. Allocation policy
4. Framework monitor
5. Migration of virtual instances
6. VM selection policy

4.3.1 Design of Cloud Environment

This phase designs the data center environment with the help of CloudSim simulator. The basic configuration of servers like types of a host, speed of processors (MIPS), number of processing elements, size of memory, bandwidth for I/O data transmission, the capacity of storage, and number of servers are placed to create a power data center. In the power data center environment, we can instantiate the virtual resources to fulfill the user's requirements. As per the user's requirements,

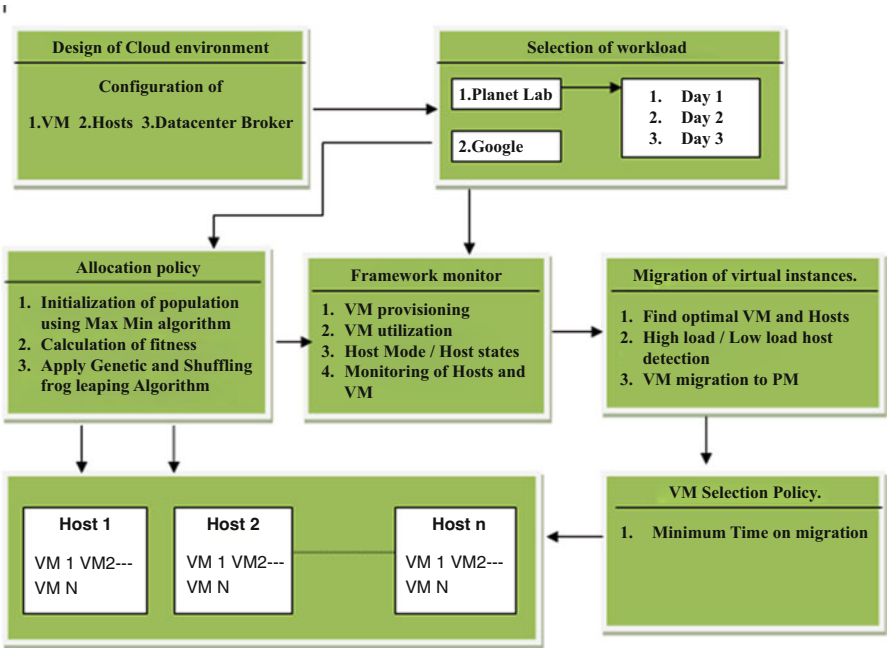

Fig. 4.1 Workflow of GESFLA

the VMs are configured and data center is initialized to experiment with our proposed and existing model.

4.3.2 Selection of Workload

(a) PlanetLab
The PlanetLab's realistic traces of VM utilization were collected for 3 days, it has 288 readings. The PlanetLab workload traces are converted into cloudlet format and are used as per user's requirements for experimental purpose and it consists of 10 days task details respectively.

(b) Google Datasets
A new "2019" trace [18] has been released by Google, which contains comprehensive Borg work scheduling data from eight different compute cluster of Google. The Google Computing Clusters (Borg cells) contains data of May 2019.The latest trace contents are an extension of the 2011 trace data [19, 20].

4.3.3 Allocation Policy

It consists of the following processes:

(i) Adaptive selection of VM.
(ii) Scheduling of Tasks.
(iii) Identification of suitable hosts based on their states and modes.
(iv) Assigning the VMs to the hosts. The processes are done by combining genetically enhanced and shuffled frog leaping algorithm as follows.

4.3.3.1 Genetic Algorithm

There are three basic genetic operations. They are selection, crossover, and mutation.

(i) The population is generated from GA, in which parents are selected randomly.
(ii) The parents are cross-bred to produce offspring in the crossover.
(iii) The children will be altered as per the policy of mutation.

In genetic algorithm, the solution obtained is called individuals, and the generations are considered as variants of an algorithm. Apart from the basic three operations of the genetic algorithm, it has some preliminary and repetitive operations such as:

(i) Initial production of the population.
(ii) Fitness function.

(i) Initial Production of the Population
The initial population includes individuals in which chromosome sizes are specified. In this method, the initial population is considered for having a better solution from "MIN-MIN" [19] and "MAX-MIN" [20] scheduling methods.

Meta Task (MT): In the data center, the series of tasks will be assigned to available resources. MT $= \{t1, t2, t3, \cdots, tn\}$.

"Min-Min Algorithm: It is based on the concept of assigning task having minimum completion time (MCT) for execution on the resource" [19]. The Min-Min algorithm is divided into two steps:

(i) The completion time of each task is calculated on each resource of the meta task.
(ii) The job with the shortest processing time is chosen and allocated to the respective resource, and the selected task is deleted from the meta task. This method continues until all the meta tasks are mapped.

Algorithm 4.1: Min-Min Algorithm

Input: Submitted task
Output: Completion time

Stage 1: *The minimum completion time of each task is calculated.*

1. *The tasks (ti) are submitted to meta task (MT).*
2. *Assign resources* R_j.
3. *Calculation of completion time* $CT_{ij} = ET_{ij} + r_j$ *is carried out.*
4. *Step 1 is ended.*
5. *Step 2 is ended.*

Stage 2: *Allocation of the resource to task ti with a least completion time.*

6. *In the MT, find the task ti with minimum completion time and the estimated resource.*
7. *Check for the task ti, which has a minimum completion time and assign to the resource Rj.*
8. *From MT, delete task ti.*
9. *Ready time of resource rj is updated.*
10. *Unmapped tasks' completion time is upgraded in MT.*
11. *Until all activities have been mapped in meta task (MT), repeat 6–10 steps.*
12. *Step 6 ends.*

"Max-Min Method. It is built on the concept of assigning a task having minimum completion time (MCT)" [20]. For resource execution, it must have the maximum completion time (fastest resource). This algorithm is divided into two steps.

(i) The completion time of each task is calculated on each resource of the meta task.
(ii) The job with the longest processing time is chosen and allocated to the respective resource, and the selected task is deleted from the meta task. This method continues until all the meta tasks are mapped.

Minimum Completion Time (MCT): Assigning each task to the available host so that the request can be completed as quickly as possible, which will yield the fastest result, which means evaluating the resource availability before allocating the job. Minimum completion time can be calculated by adding task t_i execution time, known as ET_{ij}, and resource ready time or start time, known as r_j. This can be expressed as:

$$MCT = ET_{ij} + r_j.$$

Algorithm 4.2: Max-Min Algorithm

Input: Submitted task
Output: Completion time
Stage 1: The minimum completion time of each task is calculated.

1. *The tasks (ti) are submitted to meta task (MT).*
2. *Assign resources* R_j.
3. *Calculation of completion time* $CT_{ij} = ET_{ij} + r_j$ *is carried out*

4. *Step 1 is ended.*
5. *Step 2 is ended.*

Stage 2: Allocation of the resource to task ti with a maximum completion time.

6. *In the MT, find the task ti with minimum completion time and the estimated resource.*
7. *Check for the task ti, which has a minimum completion time and assign to the resource Rj.*
8. *From MT, delete task ti.*
9. *Ready time of resource rj is updated.*
10. *Unmapped tasks' completion time is upgraded in MT.*
11. *Until all activities have been mapped in meta task (MT), repeat 6–10 steps.*
12. *The loop of Step 6 is ended.*

(ii) Fitness Function

The fitness value plays a key role in determining the individuals to create the next generation. In this work, the execution time (makespan) is considered, in which the fitness value and the fitness function are obtained through Max-Min method. To enhance resource management in the cloud environments, the utilization of resources is increased, the makespan of the task is minimized and the energy consumption of the host is also reduced. In this proposed algorithm, the GA operators, the selection, crossover, and mutation operations are applied. A set of solutions are produced; these solutions are considered to be the initial population of SFLA, and the solution which is generated will become the next population to GA. These operations are performed several times, and all the solutions are modified when the SFLA terminates the operations. In this algorithm, the early convergence is avoided by comparing the generated children with their parents. If the fitness value is improved in comparison with parents, then the children must replace the parents or it will be terminated. Based on the generation of the initial population, the fitness values are evaluated. The termination condition is verified for each iteration of the algorithm and the optimal solutions will be generated. Otherwise, the operators of GA and SFLA algorithms are applied respectively to the individuals.

4.3.3.2 Shuffled Frog Leaping Algorithm

"SFLA" [21] is a meta-heuristic strategy of optimization. It is focused on analyzing, imitating, and predicting a group of frog's behavior while looking for the position with the maximum quantity of food available. It is used to resolve several non-linear, non-differentiable, and multi-model optimization challenges. SFLA has been introduced mainly to solve the problems that occur in the field of engineering. The most distinguished advantage of SFLA is its quick convergence speed. The SFLA incorporates the advantages of both the "Particle Swarm Optimization" and the "Memetic Algorithm" (MA) [22] influenced by social behavior.

SFLA is a random search algorithm based on population, influenced by nature memetics. In the SFLA, a feasible solution population is represented by a community of frogs which is divided into multiple groups called memeplexes. Every frog executes a local search in the memeplexes. The memeplexes are allowed to combine after a specified range of memetics evolution steps, and new memeplexes are created through a shuffling phase. The shuffling processes and local search continues until the convergence constraints are fulfilled.

Algorithm 4.3: Proposed GESFLA (Genetically Enhanced Shuffling Frog Leaping Algorithm)

Input: GenesList = List of VMs and List of Cloudlets (tasks)
Output: Scheduling of tasks to VM

1. *Create an initial population*
2. *Initialize parent 1 and parent 2*
3. *Calculate fitness value of both parent 1 andparent2*
4. *Set fitness value to population list*
5. *Population list = genesList and fitnessMakespan*
6. *Apply selection operator of Genetic*
7. *Call method 1*
8. *Apply mutation operator*
9. *Calculate the fitness of offspring produced*
10. *If (fitness of child > fitness of parent)*
11. *Add the offspring into the population list*
12. *End if*
13. *Apply mutation operator*
14. *Check the fitness value of the mutated population with parent's*
15. *If (population. fitness makespan>mutationfitness)*
16. *Mutated population parent key value to be replaced*
17. *End if*
18. *Apply SFL Algorithm*
19. *Initialize frog_population list*
20. *Assign population of genetic to frog_population list*
21. *Select the population with a minimum fitness value*
22. *Create the frog list*
23. *Generate the virtual population and memeplex*
24. *Compute performance of memeplexes*
25. *Partitioning frog population into sub-memeplexes*
26. *Shuffle the memeplexes*
27. *Calculate the number of frogs for each submemeplex*
28. *Calculate the memetic evolution within each memeplex*
29. *Find execution time and makespan of sub-memeplex*
30. *Schedule the tasks to VMs*
31. *Calculate execution time and completion time of each memeplex*

Method 1: Selection Operator of Genetic

Input*: Initial population*
Population list = number of task, number of VM's
Pop size = number of parents
Output*: fittest chromosome*

1. *Assign pop size*
2. *While initial population size <pop size*
3. *Do*
4. *Create pop size random integers*
5. *Compute fitness*
6. *Spin the roulette wheel = pop size*
7. *If then*
8. *Choose the chromosome*
9. *End if*
10. *End while*

4.3.4 Framework Monitor

It verifies and examines the behavior of the data center. The host usage and resource management for the user's service level agreement are retained, so that it should not exceed the threshold limit. In case the hosts are in the state of high or low loaded [23, 24], it begins to move the VMs from the current host to another appropriate host. This constantly monitors the host states, configurations, and data center activities.

Consider number of PMs = "n" and number of VMs = "m" to execute the tasks in a data center, the use of PM (PMu) will be evaluated based on the allocated number of VMs in it as shown in Eq. (4.1).

$$\sum_{i=1}^{n} \mathrm{CVM}_{ij} \mathrm{PM}_{u} = \mathrm{CPM}_{j} \tag{4.1}$$

$i = 1$ to m, $j = 1$ to n.

Where CVM_{ij} is computing power of CPU (MIPS*Pes) of ith Virtual Machine (VM_{ij}) on jth PM.

CPM_j is the processing capability of the jth PM (PM_j) CPU (MIPS*Pes).

The PM in "active" mode has the following threshold limit to define the computational resource states.

(i) Upper threshold limit of PM >75% – High state.
(ii) 60%< threshold limit of PM ≤75% – Balanced state.
(iii) 40%≤ threshold limit of PM ≤60% – Imbalanced state.
(iv) Lower threshold limit of PM <40% – Low state.

Algorithm 4.4: VM Allocation

Input: List of VMs, List of PMs
Output: Allocation of VM to PM

1. *Check for host state: active or idle*
2. *If (host== idle)*
3. *sleep*
4. *else*
5. *check for suitability for placing the VM*
6. *if (Host utilization=Over utilized || Host utilization== Underutilized)*
7. *Host utilization = over utilized*
8. *Go to step 10*
9. *Host utilization = under utilized*
11. *Go to step 22*
12. *End if*
13. *If (hostUtilization> 0.75)*
14. *Host = highloaded;*
15. *host.setHostStates("High");*
16. *host.setHostUtilization(hostUtilization);*
17. *End if*
18. *else if (hostUtilization<= 0.75 &&hostUtilization> 0.60)*
19. *Host =BalancedHost*
20. *host.setHostStates("Balanced");*
21. *host.setHostUtilization(hostUtilization);*
22. *End if*
23. *else if (hostUtilization<= 0.60 &&hostUtilization>0.40)*
24. *Host = ImbalancedHost;*
25. *host.setHostStates("Imbalanced");*
26. *host.setHostUtilization(hostUtilization);*
27. *End if*
28. *else if (hostUtilization<= 0.40 &&hostUtilization> 0.0)*
29. *host.setHostStates("Low");*
30. *host.setHostUtilization(hostUtilization);*
31. *LowLoadedHost.add(host);*
32. *End*
33. *else*
34. *host.setHostUtilization(hostUtilization);*
35. *host.setHostMode("Sleep");*
36. *End*
37. *Allocate VMs to PMs*
38. *End*

4.3.5 Migration of Virtual Instances

VM migration [25] plays a key role in virtualization, allowing for quick migration of a VM from one physical host to another. VM migration can prove helpful in different situations like (i) balancing the load, (ii) maintenance of DC, (iii) energy management, and (iv) the failures of VMs.

4.3.5.1 High Loaded Host Detection

If a host's CPU consumption drops below the threshold, then all VMs must be transferred from the current host to another host and to minimize idle power usage. The current host should be switched to the sleep mode. If the consumption crosses the higher limits of the threshold, it is required to transfer such VMs from the existing host to minimize resource usage and to avoid SLA violations [26].

4.3.5.2 Low Loaded Host Detection

In this case, the system has a list of underloaded hosts [27]. Next, the system needs to verify if VMs can be transferred to the other servers. Therefore, the proposed algorithm checks the possibility of reorganizing all VMs to other active hosts before commencing the live migrations. A host must fulfill three conditions for accepting any VM:

1. It should not be under pressure: Once the host is in the migration process, it cannot be approached for another migration.
2. It should have sufficient resources for VM: In this situation, the capacity of the CPU, memory, and bandwidth are considered, it should not exceed the capacity of the PM.
3. The VM should not be overloaded: After moving the VMs to an imbalanced or idle host and if migration overloads the imbalanced host, the migration cycle will be aborted to the specific host.

If the other active hosts accept all VMs, the host will be shifted to sleep mode and its VMs will be included in the transfer list; or else the host will remain operational. This cycle is repeated iteratively, for all under loaded hosts.

Algorithm 4.5: VM Placement

Input: List of VMs
Output: Placing the VM to PM

1. Create VM migration list
2. *Sort VM according to their MIPS*
3. *Find the host for the VM migration*

```
4. if (migratedHost != null)
5. go to step 9
6. migrate.put("host," migratedVm);
7. End if
8. return
9. Get host list
10. for (Host host : getHostList())
11. if (host.isSuitableForVm(vm))
12. Calculate host utilization
13. utilization=((vm.getMips*vm.getNumberOfPes())/host.getTotalMips())
14. result = host.vmCreate(vm);
15. if (utilization >= 0.75)
16. if (utilization != 0 && utilization > 0.75)
17. Migration of VM to the host
18. Get VM id and host Id
19. Calculate host utilization
20. End if
21. End if
22. else
23. allocatedHost = host;
24. host.setHostUtilization(utilization);
25. Get the allocation of VM Id to the host
26. Get allocated host Id
27. return allocated host;
28. End for
29. Get the Vmsto Migrate FromUnderUtilized Host
30. for (Host HOST : host)
31. HOST.getId() &HOST.getHostStates());
32. Get the VM migrated list
33. for (Vmvm : HOST.getVmList())
34. if (!vm.isInMigration())
35. vmsToMigrate.add(vm)
36. End if
37. return vmsToMigrate;
38. End for
```

4.3.6 VM Selection Policy

When a server is overloaded, the next step is to select the VM's and transfer from the server. In this section, VM selection policy is discussed. After the VM is chosen for migration, the server is tested for over-loaded condition. If the server is still deemed

as overloaded, then again the VM selection policy will be implemented. The VM selection policy is discussed in Sect. 4.3.6.1.

4.3.6.1 Minimum Time on Migration (MTM)

The MTM strategy migrates a VM (VM) that takes less amount of time to accomplish a migration in comparison with other VMs assigned to the server. "The transfer time is measured as the amount of the RAM used by the VM, separated by the available network bandwidth of the server" [28]. V_k is a collection of VMs assigned to server "k." The MTM strategy discovers a VM (b) that matches the conditions shown in Eq. (4.2).

$$\begin{array}{l} \mathrm{RAM}_u(b)/\mathrm{NET}_i \\ \mathrm{vm} \in V_k \,|\, \forall b \in V_k \end{array} \tag{4.2}$$

Where,

$\mathrm{RAM}_u\ (b)$ = percentage of RAM presently used by the VM (b)
NET_i = spare bandwidth for the server k

The transfer time is calculated using the formula given below. The total bandwidth available for the Host is 1 GB. Half of the network bandwidth is used for communication among the VMs and another half is used for the VM migration.

1. VMs Transfer Time
The transfer time of the VM on the PM is calculated using the formula as follows:

$$\text{Transfer time} = \sum_{i=1}^{n} \frac{1}{2} * \frac{\mathrm{VM_{RAM}}}{\mathrm{Host_{BW}}}$$

Where,

Transfer Time: This refers to the time that needed to migrate the task to the candidate VM.
$\mathrm{VM_{Ram}}$ = Total available ram of VM.
$\mathrm{Host_{BW}}$ = Total network bandwidth = 1 GB/s.

2. Utilization Ratio
The resource utilization of each host is calculated using the formula as follows:

$$\text{Host utilization ratio} = \text{(VM requested MIPS) / (Total Host MIPS)}$$

$$\text{Memory utilization ratio} = \text{(VM requested RAM) / (Host Total RAM)}$$

3. Reduction of Power Consumption
In our model, the number of VMs = N, number of PMs = M, and set of resources = R. The Y_j variable checks if the PM_j is operational and the X_{ij} variable checks if VM_i assigned to PM_j. Our goal is to minimize a data center's energy consumption, based on the formula as shown in Eq. (4.3).

$$P_j = \left(P_j^{\text{active}} - P_j^{\text{idle}}\right) \times U_j^P + P_j^{\text{idle}} \tag{4.3}$$

Where, U_j^P is the usage of the CPU ($U_j^P \ \varepsilon$ [0, 1]), and P_j^{active} and P_j^{idle} are the average power values while the jth PM is active and idle.

The consumption of total energy is shown in Eq. (4.4).

$$\text{Min} \sum_{j=1}^{M} P_j^{\text{PM}} = \sum_{j=1}^{M} Y_j \left(P_j^{\text{active}} - p_j^{\text{idle}}\right) \times \sum_{i=1}^{N} \left(X_{ij} \cdot R_{i,1}{}^{\text{vm}}\right) + P_j^{\text{idle}} \tag{4.4}$$

Where,
$R_{i,1}{}^{\text{vm}}$-is a set of CPUs that are needed by VM_i
Virtual Machines (VMs) = N
Physical Machines (PMs) = M,
The total number of resources = R.
The Y_j variable determines whether PM_j is operational
The X_{ij} variable determines whether VM_i has been allocated to PM_j.

In the VM formula, R_i, VM is a range of CPUs that VM_i requires.

4.4 Result and Analysis

The suggested algorithm is designed to be implemented through the conceptual framework. Minimizing resource usage and optimizing the transfer time is the primary focus of our proposed algorithm.

4.4.1 Experimental Results

Experimental Setup: In heterogeneous environment, we have used various combinations of VMs and PMs. Table 4.1 shows the different types of PMs and VMs combinations for conducting detailed experiments. To check the proposed algorithm results, we have used the Cloudsim simulator [29]. Java language is used to implement our proposed GESFL algorithm.

The simulation is carried out using PlanetLab [28] datasets, which is a part of the CoMon [28] initiative. CPU usage data was taken from 500 different

Table 4.1 Configuration of simulation

Factor	Value
Host configuration	HP ProLiant ML110 G4 servers MIPS = 1860 PES = 2 RAM = 4096 HP ProLiant ML110 G5 servers MIPS = 2660 PES = 2 RAM = 4096 BW = 1 Gbit/s STORAGE = 1 GB
Number of hosts	800 400 = HP ProLiant ML110 G4 servers 400 = HP ProLiant ML110 G5 servers
VM configuration	1. Extra-Large [MIPS = 2500, PES = 1, RAM = 1740] 2. Medium [MIPS = 2000, PES = 1, RAM = 1740] 3. Small [MIPS = 2000, PES = 1, RAM = 870] 4. Micro [MIPS = 500, PES = 1, RAM = 613] BW = 1 Gbit/s = for all 4 types of VM.
Google cluster datasets	29 days
PlanetLab-workload	10 days

locations around the world which are running on the server from thousands of VMs. Therefore, every trace has $(24 \times 60)/5 = 288$ entries. CPU usage was taken in intervals of 5 min. The server with more cores are chosen mainly to simulate a large number of servers and to analyse the impact of consolidating VM. It is advantageous to simulate less powerful CPUs, as fewer tasks are needed for server overloading. Therefore, to check the resource management algorithms designed for multi-core processor architectures, dual-core CPUs are sufficient.

A dataset of Google Cluster Trace is published by Google in May 2019 and contains about 30 days of cell results. A collection of multiple machines sharing an eight cluster management system is defined by each cell. Each trace job involves one or more tasks that may contain many processes to be run on a single machine for each task. The trace data includes the following tables: Table of Machine Events, Table of Machine Attributes, Table of Instance Events, Table of Collection Events, and Table of Instance Usage.

In instance usage table, the *cpu_usage_distribution* and *tail_cpu_usage_distrib ution* vectors provide detailed information about the distribution of CPU consumption with 5 min intervals. These details are used to carry out the simulation.

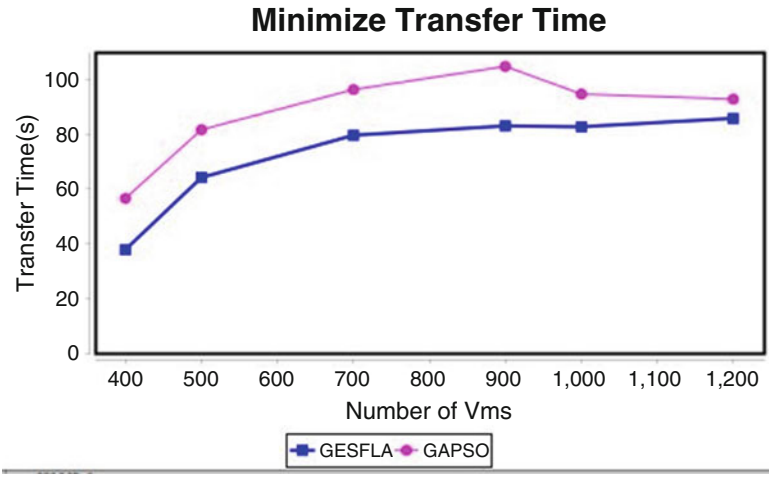

Fig. 4.2 VM transfer time (Google datasets)

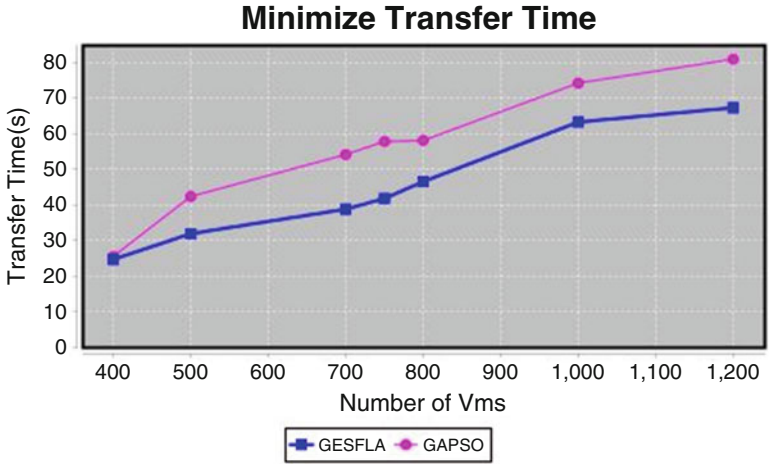

Fig. 4.3 VM transfer time (PlanetLab)

Further, the GESFL algorithm transfer time is shown in Figs. 4.2 and 4.3 is relatively smaller in comparison with GAPSO. If the load is balanced equally among VMs, then the CPU is also utilized efficiently as shown in Figs. 4.4 and 4.5. The energy usage is depicted in Figs. 4.6 and 4.7.

To determine the efficiency of our proposed GESFL algorithm, various combinations of VMs and PMs are compared with GAPSO in the context of resource utilization, time of migration, and energy consumption. GESFLA's migration time is calculated for the Google datasets and PlanetLab datasets as shown in Table 4.2.

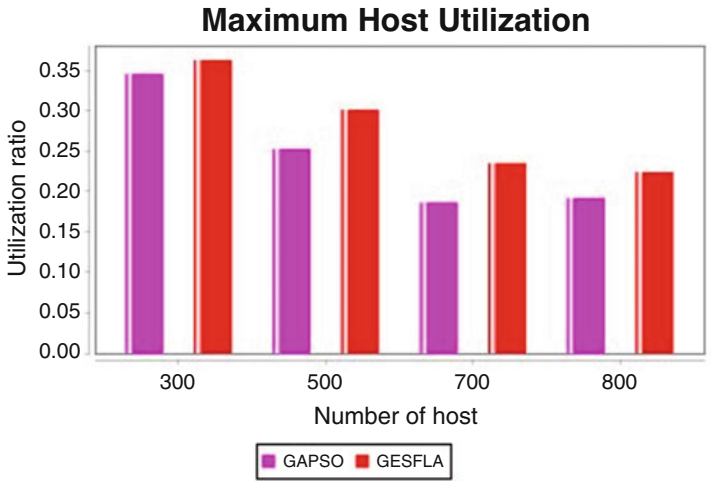

Fig. 4.4 Utilization of the host (Google datasets)

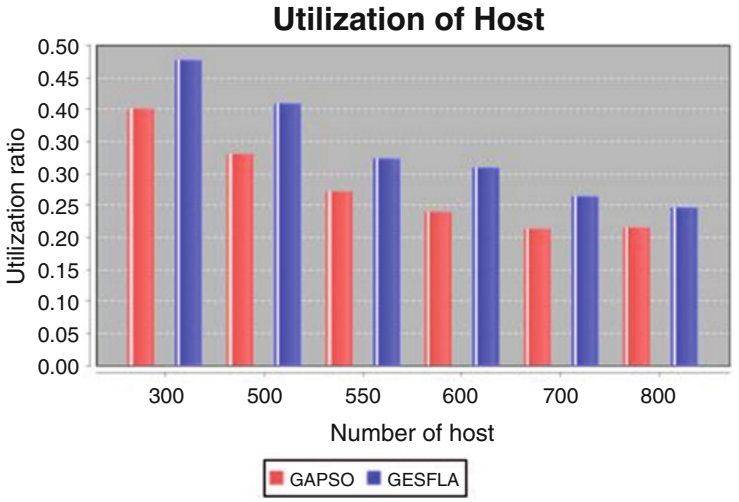

Fig. 4.5 Host utilization (PlanetLab)

The improvement in the maximum utilization of our suggested algorithm is calculated as shown in Table 4.3. The algorithm provides better outcomes for power usage over the GAPSO algorithm, due to the usage of the VM transfer strategy to turn off under loaded PMs at the data center. Thus, the CPU usage of PMs is improved by reducing the number of VMs in data center. The migration of VMs is built on fixed time intervals in data center. For every 60 s, we have received the utilization status of every used PM and several VMs that are assigned to underused PMs. Next, we can turn off the underused PMs in data center by transferring the

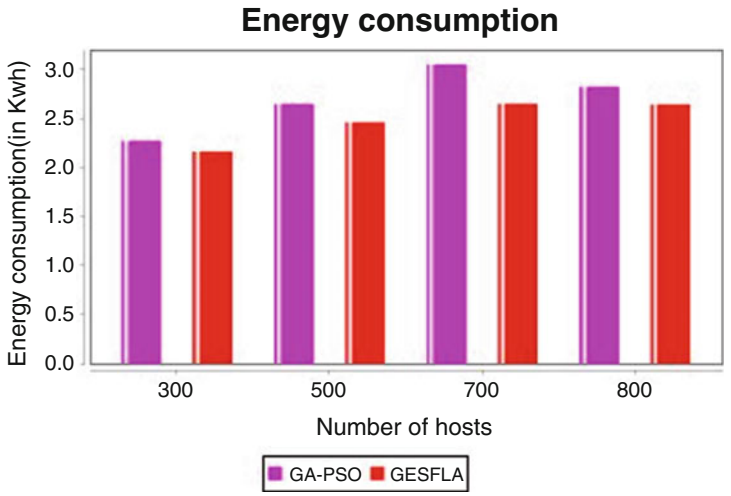

Fig. 4.6 Power consumption (Google datasets)

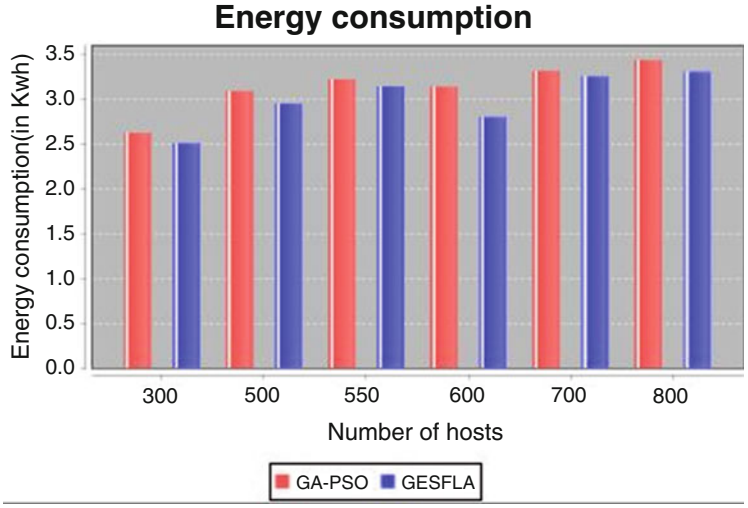

Fig. 4.7 Power consumption (PlanetLab)

VMs using the migration policy discussed in Sect. 3.6.1. Table 4.4 depicts the improvement of energy consumption of the hosts.

Since the fewer number of PMs and VMs are used, the transfer time of the proposed GESFL algorithm significantly gives better results than the GAPSO (PMs) configuration. As more PMs are switched off, data center's expense reduces. The hybrid model suggested by GESFLA has increased chromosomal fitness. The children obtained by the crossover and mutation operations are of better quality

Table 4.2 Improvement in transfer time

	Minimizing the transfer time Google datasets			Minimizing the transfer time Planetlab datasets		
Number of VMs	GAPSO (Existing)	GESFLA (Proposed)	Improvement (%) in transfer time of GESFLA with GAPSO	GAPSO	GESFLA	Improvement (%) in transfer time of GESFLA with GAPSO
400	56.53	37.79	33	25.42	24.58	3
500	82.60	63.12	23	42.28	32.56	22
700	96.28	83.203	13	54.06	47.71	12
1000	95.65	78.668	17	74.19	62.19	16
1200	77.80	60.46	22	80.88	70.22	14

Table 4.3 Improvement in CPU utilization of host

	Maximum CPU utilization of the host (Google workload)			Maximum utilization of the host (PlanetLab workload)		
No of Hosts	GAPSO (Exising)	GESFLA (Proposed)	Improvement (%) in the host usage of GESFLA with GAPSO	GAPSO	GESFLA	Improvement (%) in the host usage of GESFLA with GAPSO
300	0.345	0.3629	5	0.401	0.478	16
500	0.41	0.328	20	0.252	0.301	15
700	0.1865	0.235	20	0.213	0.264	19
800	0.191	0.223	14	0.215	0.246	12

Table 4.4 Improvement in energy consumption

	Minimizing energy consumption (Google workload)			Minimizing energy consumption (PlanetLab workload)		
No of Host	GAPSO (Existing)	GESFLA (Proposed)	Improvement (%) of GESFLA with GAPSO	GAPSO	GESFLA	Improvement (%) of GESFLA with GAPSO
300	2.64	2.45	3	2.62	2.50	4
500	2.26	2.16	4	3.09	2.95	5
700	3.05	2.64	13	3.31	3.25	12
800	2.82	2.65	6	3.43	3.30	3

and thus have achieved the optimal solution for allocating VMs. This helps in maximizing the usage of the resource by reducing execution time and transfer time.

The efficiency of GESFLA is increased because mutation and crossover operators of genetic algorithm helps to generate quality of good children. The children produced are grouped into memeplexes using SFLA. The memeplexes does the shuffling of children produced and the local search of availability of resources. This takes very less convergence time, which reduces the transfer time of VMs from one host to another.

It is necessary to consider the power consumption because it is a crucial factor for data center's [30] and also for the service providers. Minimizing power consumption [31] helps to reduce the cost of data center. We are considering under loaded hosts and enabling the host to switch to the balanced state by shutting down the under loaded host. The shuffling process of SFLA helps VMs to find the underloaded hosts and it is beneficial in maintaining the load among the hosts.

4.4.2 Analysis of Complexity of Time

The proposed GESFLA processing time is dependent upon GA and SFLA algorithms.

Consider,

"k" = number of tasks
"x" = number of PMs
"y" = number of VMs and
"s" = size of the generation' and
"C" = crossover rate of each generation

GA's time complexity is focused on the distinct operations of generation, crossover, and mutation. SFLA focuses on Frog generation, memeplex number, an update of the position, estimation of memeplex fitness, Frog's location, return the missing VMs, and delete duplicate VMs for specified intervals. GAPSO-based migration of VM needs O (nlogn) processing time. GESFLA = O (GA) + O (SFLA). As a result, the time complexity of GESFLA = O (logn) is equal to O (n: k,s,m) + n logn computation.

4.5 Conclusion and Future Work

We have discussed different phases of our proposed GESFL algorithm. The algorithm is tested for Google cluster trace datasets as well as real-time workload traces from the PlanetLab. The suggested approach used threshold values for VM migration and load on the server is reduced by frequent monitoring of all the VMs. The experimental results indicate that GESFLA framework efficiency is better than the GAPSO algorithm. The proposed algorithm increases the performance of data

center by minimizing the transfer time by 17%, maximizing resource utilization around 14–16% and energy consumption is reduced in comparison with the existing algorithm by 6%. The limitation of our proposed algorithm is that we have listed only four combinations of VMs and PMs which can be enhanced with more combinations. In the future, the algorithm can be applied for actual data centers in real-time.

References

1. H. Priyanka, Analytics of application resource utilization within the virtual machine. Int. J. Sci. Res. **5**(4), 1690–1693 (2016)
2. H. Hajj, W. El-Hajj, M. Dabbagh, T.R. Arabi, An algorithm centric energy-aware design methodology. IEEE Trans. Very Large Scale Integr. Syst. **22**(11), 2431–2435 (2014)
3. F. Ramezani, J. Lu, F.K. Hussain, Task-based system load balancing in cloud computing using particle swarm optimization. Int. J. Parallel Prog., Springer **42**, 739–754 (2014)
4. L. Guo, S. Zhao, S. Shen, C. Jiang, Task scheduling optimization in cloud computing based on a heuristic algorithm. J. Networks **7**(3), 547–553 (2012)
5. H. Priyanka, M. Cherian, The challenges in virtual machine live migration and resource management. Int. J. Eng. Res. Technol. **8**(11), 5 (2020)
6. N.K. Sharma, G. Ram Mohana Reddy, Multi-objective energy efficient virtual machines allocation at the cloud data center. IEEE Trans. Serv. Comput. **12**(1), 158–171 (2019)
7. Y. Ge, G. Wei, GA-based task scheduler for the cloud computing systems, in *Proceedings of the International Conference on Web Information Systems and Mining (WISM '10)*, vol. 2, (IEEE, 2010), pp. 181–186
8. G. Giftson Samuel, C. Christober Asi Rajan, Hybrid: Particle Swarm Optimization–Genetic Algorithm and Particle Swarm Optimization–Shuffled Frog Leaping Algorithm for long-term generator maintenance scheduling. Int. J. Elect. Power Energy Syst., Elsevier **65**, 432–442 (2015)
9. E.G. Coffman Jr., M.R. Garey, D.S. Johnson, Approximation algorithms for bin packing: A survey, in *Approximation Algorithms for NP-Hard Problems*, ed. by D. S. Hochbaum, (PWS Publishing Co, Boston, 1997), pp. 46–93
10. Y. Ajiro, A. Tanaka, Improving packing algorithms for server consolidation, in *Proceedings of the 33rd International Computer Measurement Group Conference, December 2–7, 2007, San Diego, CA, USA*, (DBLP, 2007), pp. 399–406
11. A. Beloglazov, R. Buyya, Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints. IEEE Trans. Parallel Distrib. Syst. **24**(7), 1366–1379 (2013)
12. K. Dasgupta, B. Mandal, P. Dutta, J.K. Mandal, S. Dam, A genetic algorithm (GA) based load balancing strategy for cloud computing. Procedia Technol. **10**, 340–347 (2013)
13. A.-P. Xiong, C.-X. Xu, Energy efficient multiresource allocation of virtual machine based on PSO in cloud data center. Math. Probl. Eng. **2014**, 1–8 (2014)
14. W.-T. Wen, C.-D. Wang, D.-S. Wu, Y.-Y. Xie, An ACO-based scheduling strategy on load balancing in cloud computing environment, in *Ninth IEEE International Conference on Frontier of Computer Science and Technology*, vol. 6, (IEEE, 2015), pp. 364–369
15. S. Wang, Z. Liu, Z. Zheng, Q. Sun, F. Yang, Particle swarm optimization for energy-aware virtual machine placement optimization in virtualized data centers, in *Proceedings of the IEEE International Conference on Parallel and Distributed Systems*, (IEEE, 2013), pp. 102–109
16. H. Xu, B. Yang, W. Qi, E. Ahene, A multi-objective optimization approach to workflow scheduling in clouds considering fault recovery. KSII Trans. Internet Inf. Syst. **10**(3), 976–994 (2016)

17. S. Chitra, B. Madhusudhanan, G.R. Sakthidharan, P. Saravanan, Local minima jump PSO for workflow scheduling in cloud computing environments, in *Advances in Computer Science and its Applications. Lecture Notes in Electrical Engineering*, vol. 279, (Springer, 2014), pp. 1225–1234
18. C. Reiss, J. Wilkes, J.L. Hellerstein, Google cluster-usage traces: Format+ schema. Technical report at https://github.com/google/clusterdata, Google, Mountain View, CA, USA, Revised 2014-11-17 for version 2.2, Nov. 2011
19. Y. Mao, X. Chen, X. Li, Max–Min task scheduling algorithm for load balance in cloud computing, in *Proceedings of International Conference on Computer Science and Information Technology. Advances in Intelligent Systems and Computing*, ed. by S. Patnaik, X. Li, vol. 255, (Springer, New Delhi, 2012)
20. B. Santhosh, D.H. Manjaiah, A hybrid AvgTask-Min and Max-Min algorithm for scheduling tasks in cloud computing, in *International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT), Kumaracoil*, (IEEE, 2015), pp. 325–328. https://doi.org/10.1109/ICCICCT.2015.7475298
21. J. Wilkes, More Google cluster data. Google research blog (Nov. 2011). Posted at http://googleresearch.blogspot.com/2011/11/more-googlecluster-data.html
22. J. Wilkes, Google cluster-usage traces v3. Technical report at https://github.com/google/cluster-data, Google, Mountain View, CA, USA, Nov. 2019
23. H. Priyanka, M. Cherian, Efficient utilization of resources of virtual machines through monitoring the cloud data center, in *International Conference on Communication, Computing and Electronics Systems. Lecture Notes in Electrical Engineering*, ed. by V. Bindhu, J. Chen, J. Tavares, vol. 637, (Springer, Singapore, 2020), pp. 645–653
24. H. Priyanka, M. Cherian, Novel approach to virtual machine migration in cloud computing environment – A survey. Int. J. Sci. Rep. **7**(1), 81–84 (2018)
25. L. Weining, F. Ta, Live migration of virtual machine based on recovering system and CPU scheduling, in *6th IEEE joint International Information Technology and Artificial Intelligence Conference, Piscataway, NJ, USA*, (IEEE, 2009), pp. 303–305
26. S.B. Melhem, A. Agarwal, N. Goel, M. Zaman, Markov prediction model for host load detection and VM placement in live migration. IEEE Access **6**, 7190–7205 (2017)
27. A. Kishor, R. Niyogi, Multi-objective load balancing in distributed computing environment: An evolutionary computing approach, in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, (Association for Computing Machinery, Brno, 2020), pp. 170–172
28. A. Beloglazov, R. Buyya, Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers, in *Concurrency and Computation: Practice and Experience*, vol. 24, No. 13, (Wiley Press, New York, 2011). https://doi.org/10.1002/cpe.1867
29. R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F. de Rose, R. Buyya, CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw. Pract. Exp. **41**(1), 23–50 (2011)
30. J. Singh, J. Chen, Optimizing energy consumption for cloud computing: A cluster and migration based approach (CMBA), in *Proceedings of the 2019 3rd International Conference on Computer Science and Artificial Intelligence*, vol. 22, No. 11, (Association for Computing Machinery (ACM), 2019), pp. 28–32
31. S. Rahman, A. Gupta, M. Tornatore, B. Mukherjee, Dynamic workload migration over backbone network to minimize data center electricity cost. IEEE Trans. Green Commun. Netw. **2**, 570–579 (2018)