Multi-Objective Energy Efficient Virtual Machines Allocation at the Cloud Data Center

Neeraj Kumar Sharma and G. Ram Mohana Reddy, *Senior Member, IEEE*

Abstract—Due to the growing demand of cloud services, allocation of energy efficient resources (CPU, memory, storage, etc.) and resources utilization are the major challenging issues of a large cloud data center. In this paper, we propose an Euclidean distance based multi-objective resources allocation in the form of virtual machines (VMs) and designed the VM migration policy at the data center. Further the allocation of VMs to Physical Machines (PMs) is carried out by our proposed hybrid approach of Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) referred to as HGAPSO. The proposed HGAPSO based resources allocation and VMs migration not only saves the energy consumption and minimizes the wastage of resources but also avoids SLA violation at the cloud data center. To check the performance of the proposed HGAPSO algorithm and VMs migration technique in the form of energy consumption, resources utilization and SLA violation, we performed the extended amount of experiment in both heterogeneous and homogeneous data center environments. To check the performance of proposed HGAPSO with VM migration, we compared our proposed work with branch-and-bound based exact algorithm. The experimental results show the superiority of HGAPSO and VMs migration technique over exact algorithm in terms of energy efficiency, optimal resources utilization, and SLA violation.

Index Terms—Energy efficiency, data center, genetic algorithm, particle swarm optimization, virtual machine, euclidean distance, physical machine, resources utilization

1 Introduction

The increasing demand of cloud services by the small and large scale industries, the size of the data centers is continually increasing now a days. To meet the ever increasing demand of the customers, more number of servers are needed at the data center. These data centers require more cooling devices in order to keep the data center at a specified temperature resulting in more energy consumption and CO_2 emission [1]. The cost of physical resources (CPU, memory, storage, etc.) of the data center is 15 percent of the total operating cost, and the energy consumption cost is 45 percent of the total operating cost [2]. Hence, all these said factors gave importance to find the energy efficient resources allocation at the cloud data center in the direction of not only reducing the operating cost but also making it as green data center.

The energy consumption of the data center is mainly due to the under utilization of the PMs. The idle PM consumes more than 50 percent of the energy it consumes in fully loaded condition [3]. For improving the resources utilization of the PM, number of VMs created on a PM using the virtualization software (Hypervisor, KVM, Containers etc.). The user requested on demand VMs allocation problem is widely known as a combinatorial optimization problem, it is also known as a multi-dimensional variable size bin-packing problem where items are the VMs and the bins are the PMs [4]. Due to the large number of PMs present in the data center, the specified VMs allocation problem is related to the NP-hard/NP-Complete complexity class [5]. Finding an optimal solution of the specified VMs allocation problem with multi-objective approach will thus create a lot of challenges to the researchers. The complexity of the multi-objective VMs allocation problem is generally defined in two different ways. When the similar types of VMs and PMs are considered in the data center (homogeneous environment), then the reduction of resources wastage can also reduce the energy consumption.

Further, when different types of VMs and PMs are considered in the data center (heterogeneous environment), then the reduction of resources wastage may not guarantee similar reduction of energy consumption [6]. Therefore, one global optimal solution is required in limited time frame, where we can achieve both the energy efficiency and minimization of the resources wastage in homogeneous and heterogeneous data center environment. Further we need to migrate VMs from one PM to another PM in such a manner that we can save the energy consumption by switching-off under utilized/ideal PMs without violating the SLA. The VMs allocation in the cloud data center is classified into three categories: i) Static VM allocation technique which allocates the VMs on PMs for a specified amount of time. After the expiry time of VMs, the algorithm will be re-executed and thus destroys the expired VMs; ii) Semi-Dynamic VMs allocation technique by which the VMs allocation algorithm is executed at a specified amount of time; iii) Dynamic VMs allocation technique which deals with on-demand resources allocation based on future predictions of the workload. Each of these techniques has some merits and demerits in terms of SLA violation, amount of energy consumption, and the resources utilization.

Hence, in this paper, we mainly focus on the key goals of multi-objective VMs allocation on PMs, and VM migration

• *The authors are with the Department of Information Technology, National Institute of Technology Karnataka, Mangalore, Karnataka 575025, India. E-mail: {neeraj16ks, profgrmreddy}@gmail.com.*

Manuscript received 10 Apr. 2016; revised 10 July 2016; accepted 15 July 2016. Date of publication 29 July 2016; date of current version 6 Feb. 2019.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSC.2016.2596289

at a cloud data center in terms of energy efficiency, minimization of resources wastage, and avoiding SLA violation. We propose an hybrid algorithm referred to as a HGAPSO based on Genetic Algorithm (GA), Particle Swarm Optimization (PSO), and euclidean distance in achieving the optimal point of energy efficiency and resource utilization. Further we design a VM migration policy for saving energy and avoiding SLA violation in the data center. The hybridization of evolutionary algorithms has been widely used by different researchers for different applications [5], [7], [8]. In our proposed hybrid approach of VMs allocation at the cloud data center, we consider the IaaS cloud model. Cloud services provider will provide different types of VMs to customers on rent basis for a specified period of time. On the basis of amount of VMs purchased by the customers and the application deadline, we define a strong SLA policy w.r.t. to response time. The main objective of this proposed work is to formulate the VMs allocation problem as a multi-objective multi-dimensional combinatorial optimization problem, and thus allocates the multiple instances of users requested VMs on minimum number of energy efficient PMs in such a manner that we can reduce both the energy consumption and the resources wastage at the cloud data center.

The core idea of proposed hybrid approach of HGAPSO is to apply the GA for the generation of initial VMs allocation on PMs and then apply the PSO for the improvement of the initial solution and thereby getting the near global optimal solution of the VMs allocation problem.

The key/main contributions of this paper are as follows:

1) A multi-objective, multi-constraints VMs allocation at the data center using proposed HGAPSO.
2) The calculation of energy consumption and resource utilization in both homogeneous and heterogeneous environments of the cloud data center.
3) To design SLA violation policy in terms of purchased resources, response time, and deadline time of an application.
4) To design energy efficient SLA aware VMs migration policy.

The rest of the paper is organized as follows. Section 2 deals with the related work on the energy efficient VMs allocation. Section 3 describes the proposed HGAPSO algorithm and VM migration policy. Section 4 discusses the Experimental setup, Results and Performance Evaluation of HGAPSO. Finally, Section 5 deals with Conclusion and Future work.

2 Related Work

Previous research on energy savings in the cloud data center is divided into two different categories: i) single server level; and ii) multiple server level or grid level.

2.1 Single Server Level Energy Saving Techniques

Ge et al. [9] and Hsu et al. [10] suggested Dynamic voltage frequency scaling (DVFS) based approaches for energy efficiency of a single server by restricting the use of CPU utilization well below the upper threshold value of CPU utilization. The time-out based approach proposed by Kveton et al. [11] kept the CPU on lower power state under no work load condition beyond a certain time-out threshold. The main limitation of this approach is that it does not switch CPU to the lower state until the time-out period has passed; hence, it will not save the energy during this time period. Wu et al. [12] reduced the energy consumption of the cloud data center using DVFS. Their approach was based on DVFS that takes the task on priority basis and scheduled on minimum amount of resources. By this way, the overall utilization of the data center is increasing and resulting in lower energy consumption at the data center. The disadvantages of their work are lower priority task suffered by lower response time and there may be a chance of SLA violation.

2.2 Multiple Server Level Energy Saving Techniques

The main focus in this approach is to minimize the energy consumption of the data center by using minimum number of energy efficient PMs for the VMs allocation while switching-off unused PMs at the data center. To solve VMs allocation problem, Ajiro et al. [13], and Man et al. [14] used heuristic approaches, such as First Fit, Best Fit respectively. But heuristic approaches will not provide global optimum solution and Best Fit heuristic approach is not scalable in nature due to the long convergence time, and the other limitation of their suggested work is based on single objective function. Beloglazov et al. [15] suggested a Modified Best Fit Decreasing (MBFD) algorithm by first sorting the VMs in the decreasing order and PMs in the increasing order on the basis of their processing capacity. After sorting of VMs and PMs, allocation of VMs on PMs is done by using First Fit Decreasing (FFD). The limitations of their work are: single objective based VMs allocation and MBFD which is not in scalable in nature when large numbers of requested VMs are arrived at the data center.

Some other works used similar type of VMs allocation problem with different algorithms. Other researchers used bio-inspired and nature-inspired algorithms, such as GA, PSO, CSO, etc. for VMs allocation at the cloud data center. Xiong et al. [16] designed energy efficient VMs allocation problem using PSO. The limitation of this work is that authors considered single type of VMs. Gao et al. [17] suggested a multi-objective ant colony based VMs allocation at the cloud data center. The limitation of this work is that authors considered the homogeneous data center in the form of VMs and PMs. Wang et al. [18] suggested the energy efficient VMs placement in the data center using PSO. The limitation of their work is that non-energy aware random reallocation of VMs after velocity changes takes a lot of iterations and gave a non-optimal solution. Xu et al. [19] developed the multi-objective based VMs placement in the cloud data center and thereby minimizing the energy consumption and the resources wastage using GA and fuzzy logic techniques. The problem with this approach is that some infeasible VMs are reallocated randomly to the PMs after crossover and mutation operations and this has resulted an inferior quality solution in the next generation of chromosomes.

Except approximation and bio-inspired algorithmic approaches, several predictive frameworks [20], [21] were proposed to reduce the number of switched-on PMs. Rather than reserving the VMs for each application all the time,

TABLE 1
Summary of Related Work

Author	Methodology	Limitations
Garg et al. [24]	Adjusting operating frequency of the CPU using DVFS.	Deals with single server not cluster level.
Shojafar et al. [25]	Energy saving adaptive computing using DVFS in grid environment.	During no work load condition the CPU still consumes the power.
Vinh et al. [26]	Load prediction using historical data, deals with dynamic power saving.	Accurate load prediction from previous data is not possible, SLA violation with reference to user's response time.
Ghribi et al. [27]	Energy efficient resources allocation and migration with both static and dynamic power saving.	Energy calculation in non-heterogeneous environment, during VMs migration SLA violation may occur.
Karthik et al. [28]	Multi queue job scheduling, saves dynamic power.	Not dealing with static power.

Nguyen [22] dynamically adjusted the number of switched-on PMs by predicting the workload of the data center. Xio et al. [23] suggested the dynamic resources allocation using VMs in cloud computing. The major disadvantage of this load prediction based VM consolidation approach is that it suffers from SLA violation if the load is not predicted appropriately. Hence, in the existing literature survey, several researchers either worked at the single server level or multiple server level (grid level) homogeneous cloud environment. Table 1 summarizes the related works.

3 Proposed HGAPSO Algorithm

3.1 Energy Consumption Model

Except the cooling devices, most of the data center energy is consumed by the CPU [29]. A single CPU energy saving technique such as Dynamic Voltage Frequency Scaling is based on the two states of the CPU: idle state and workload state. In idle state (no workload condition), the Operating System switches-off some of the internal components of the CPU, and also reduces the CPU clock frequency; hence CPU works in a minimum frequency mode. In the case of workload state, the energy consumption of the CPU depends on the amount of workload applied to the CPU, and utilization rate of CPU. Hence, the DVFS energy saving technique works between two clock frequency modes (f^{min} and f^{max}). In our proposed work, we consider all the PMs based on inbuilt DVFS technique and the power and energy consumption of a PM is based on the approach of Minas and Elliso [30]. The power consumption (P_j) and the energy consumption (E_j) (during time duration of t_1 to t_2) of (pm_j) is described as follows:

$$P_j = ([P_j^{max} - P_j^{min}])u + P_j^{idle} \tag{1}$$

$$E_j = \int_{t=t_1}^{t=t_2} P_j dt, \tag{2}$$

where P_j^{max} and P_j^{min} are the maximum and minimum power consumptions of pm_j respectively; u is the rate of CPU utilization between 0 and 1; E_j is total energy consumption of pm_j during time duration of t_1 to t_2.

If a data center has 'm' number of PMs, then the total energy consumption of a data center (DC^{energy}) is defined by Eq. (3). The resource (mips, ram, storage) utilization of pm_j (u_j^d) is defined by Eq. (4). The average resource utilization of the data center (DC^u) over time duration of t_1 to t_2 is defined by Eq. (5)

$$DC^{energy} = \sum_{j=1}^{m} E_j \tag{3}$$

$$u_j^d = \frac{\sum_{i=1}^{n} a_{ij} * vm_i^d}{pm_j^d} \quad \forall d \in \{mips, ram, storage\} \tag{4}$$

$$a_{ij} = \begin{cases} 1 & if \quad vm_i \quad allocated \quad to \quad pm_j \\ 0 & else \end{cases}$$

$$DC^u = \int_{t=t_1}^{t=t_2} \frac{\sum_{j=1}^{m} u_j^{mips} + \sum_{j=1}^{ram} u_j^{ram} + \sum_{j=1}^{m} u_j^{storage}}{|d| \sum_{j=1}^{m} b_j} dt$$
$$\sum_{j=1}^{m} b_j > 0 \quad \text{and} \quad |d| = 3, \tag{5}$$

where 'b_j' is a binary variable indicating whether pm_j is used for VMs allocation or not. The value of $b_j = 1$ if pm_j is used for the VM allocation otherwise it is 0; $|d|$ = 3 is defined for the number of resources such as mips, ram, storage.

3.2 Multi-Objective VMs Allocation

Let us consider a data center with 'n' number of VMs and 'm' number of PMs. Allocation of VMs on PMs at the data center is carried out in such a manner that it gives a global optimal solution in the form of VMs allocation. The global optimal placement solution minimizes the energy consumption min $p = \sum_{j=1}^{m} P_j$ as-well-as also maximizing the resource utilization max $u = \sum_{j=1}^{m} u_j^d$. Therefore VMs allocation problem is referred to as pareto optimal problem [24]. Hence to achieve two conflict objectives (minimization and maximization) we consider the euclidean distance based objective function and thus obtaining an optimal solution for both minimizing the energy consumption and maximizing the resources utilization at the cloud data center.

The other distance metrics like Hamming distance (it measures the distance between two strings bit by bit), Mahalanobis distance (it measures the distance between a point 'P' and distribution 'D'), Haversine distance (it gives the distance between two points on a sphere) etc. are not useful and feasible for our proposed work. Hence the considered euclidean distance gives the minimum distance between two points in 'd' dimension-space. Further euclidean distance in this proposed work is not indicating the distance between the servers in the cloud data center, but euclidean distance based objective function is used for selecting the best PM for the VMs allocation. This can be achieved by calculating an optimal point between energy consumption and resource utilization. Therefore euclidean distance is the feasible and the best choice for our proposed work. Further the current resources utilization of PM_j (u_j^d) between 0 to 1. Thus we use the normalized value of power consumption between 0 to 1 by dividing the current power consumption of PM_j (P_j) by their maximum power consumption value (P_j^{max}) value

$$min \quad f = \sum_{j=1}^{m} \sqrt{\left(\frac{P_j}{P_j^{max}}\right)^2 + (u_j^d - 1)^2} \tag{6}$$

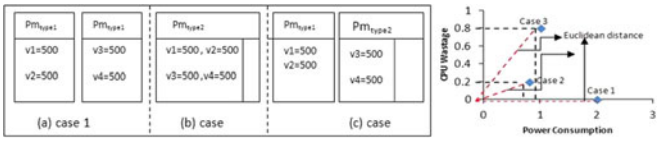


Fig. 1. VM allocation scenario.

$$vm_i^{pe} * a_{ij} \leq pm_j^{pe} \quad \forall j \in \{1, 2, 3, \ldots, m\} \tag{7}$$

$$\sum_{i=1}^{n} vm_i^{mips} * a_{ij} \leq pm_j^{mips} \quad \forall j \in \{1, 2, 3, \ldots, m\} \tag{8}$$

$$\sum_{i=1}^{n} vm_i^{ram} * a_{ij} \leq pm_j^{ram} \quad \forall j \in \{1, 2, 3, \ldots, m\} \tag{9}$$

$$\sum_{i=1}^{n} vm_i^{storage} * a_{ij} \leq pm_j^{storage} \quad \forall j \in \{1, 2, 3, \ldots, m\}. \tag{10}$$

The multi-objective minimization function based on the euclidean distance gives the optimal solution which is described by the following example.

Example. Let us consider two different types of PMs at the data center. The type1 PM consists of one processing element of 1,000 mips and maximum power consumption of (P_{type1}^{max}). The type2 PM has two processing elements of 2,500 mips each and maximum power consumption of (P_{type2}^{max}). From the specification of Intel Xeon processors available on Intel's web site [30], the power consumption is about 30 percent more on advance CPUs. Hence on the basis of power model, the inequality conditions $P_{type1}^{max} < P_{type2}^{max}$ and $2 * P_{type2}^{max} > P_{type1}^{max}$ are satisfied. Further we consider four VMs requested at the data center with one processing element of 500 mips each. Hence there are 16 different VM allocations possible at the data center. Out of these 16 combinations, we consider most power efficient feasible allocations of VMs. The allocation of VMs to the data center is shown in Fig. 1 using three cases. The function value of case two (f_2) is less as compared to function value of case three (f_3) such as ($f_2 < f_3$) due to the wastage of mips is more in case 3 as compared to case 2. Further function value of case 1 (f_1) is more as compared to case 2 even though the wastage of mips in case 1 is 0 percent because of power inequality condition. Hence the minimum optimization function value f_2 gives the optimal placement of VMs to the data center.

TABLE 2
Notations and Definitions

Notation	Definition
vm_i^{pe}	Number of processing elements requested by vm_i
pm_j^{pe}	Number of processing elements in pm_j
vm_i^{mips}	Amount of mips requested by vm_i
pm_j^{mips}	Amount of MIPS at pm_j
vm_i^{ram}	Amount of ram requested by vm_i
pm_j^{ram}	Amount of ram at pm_j
$vm_i^{storage}$	Amount of storage requested by vm_i
$pm_j^{storage}$	Amount of storage at pm_j
u_j^d	Resource utilization of a pm_j

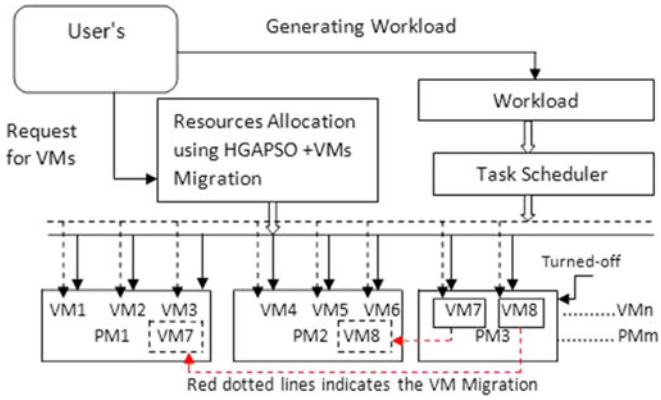


Fig. 2. Flow chart of VMs allocation and task scheduling.

The multi-objective minimization function is described by Eq. (6). The constraints satisfaction of different resources such as processing elements, MIPS, RAM, and STORAGE, are defined by Eqs. (7) to (10) respectively. The mathematical notations used in Eqs. (1) to (10) are described in Table 2.

Fig. 2 shows the flow chart of the VMs Allocation and Task Scheduling, and it consists of three phases. In the first phase, user requested VMs are allocated to the PMs at the data center using the proposed HGAPSO (Fig. 3) for a specified amount of time. In the second phase, the task scheduler will schedule the tasks to the user purchased VMs. In the third phase migrate VMs from underutilized PMs to energy efficient PMs using our proposed policy. Each user's request contains the information about the type of VMs, number of VMs, and duration of VMs. In our proposed work, we consider four different types of VMs offered by the service provider to the customer. The query of a user requested VMs for the time duration (t) is defined by R_i= $\{vm_{small}(n1), vm_{medium}(n2), vm_{large}(n3), vm_{x.large}(n4), t\}$, where $vm_{small}(n1)$, $vm_{medium}(n2)$, $vm_{large}(n3)$, $VM_{x.large}(n4)$ are the numbers of small, medium, large, and extra large types of VMs requested by a user at the cloud data center respectively.

3.3 Basic Concepts of GA, PSO

The proposed HGAPSO algorithm combines the GA and PSO. Hence, in this section, basic concepts of GA, and PSO

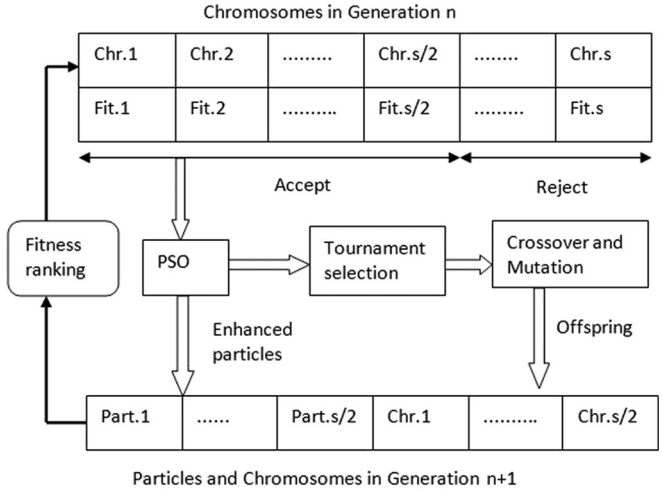


Fig. 3. Flow chart of the proposed HGAPSO algorithm.

are introduced followed by a detailed design of proposed HGAPSO algorithm in the next section.

(i) Basics of GA. GA is based on the random searching of possible solutions in the search space; each chromosome gives the possible solution in the search space. Initially, the number of chromosomes is generated randomly in the search space for generating the initial population. Based on the fitness value of chromosome, the new population is generated iteratively using genetic operations such as crossover, and mutation.

(ii) Basics of PSO. The PSO starts searching a solution using a population of particles. A population of particles (swarm) is randomly generated by an initial population. Each particle represents a feasible solution of the problem and its position is represented by a position vector $\vec{x}_i$. A swarm (group of particles) moves in the problem space, with the moving velocity of each particle defined by a velocity vector $\vec{v}_i$. At each time step 't', using the local best particle position $\vec{x}_{lbesti}$ and the global best particle position $\vec{x}_{gbest}$; both velocity vector and position of the particle for time step $(t+1)$ are calculated as follows:

$$\vec{v}_i(t+1) = w\vec{v}_i(t) + c_1 r_1(\vec{x}_{lbesti}(t) - \vec{x}_{gbest}(t)) + c_2 r_2(\vec{x}_{gbest}(t) - \vec{x}_i(t)) \quad (11)$$

$$\vec{x}_i(t+1) = \vec{x}_i(t) + \vec{v}_i(t+1), \quad (12)$$

where w is a learning coefficient between 0 and 1; c_1 and c_2 are the constants between 0 and 1; r_1 and r_2 are the uniformly distributed random numbers between 0 and 1.

3.4 Proposed HGAPSO

Fig. 3 shows the flowchart of the proposed HGAPSO algorithm. The motivation behind using the combination of GA and PSO are described as follows. The similarities between these two techniques are: both GA and PSO techniques work on a random population for searching a solution in the defined search space to solve the discrete optimization problem. In the proposed HGAPSO, the crossover operation of GA changes the placement of VMs from one chromosome to the another chromosome by migrating the VMs from one chromosome to the other chromosome. However, GA takes more convergence time and its performance is very poor when the solution space is large and the fitness values of the parent chromosomes are low. The other reason for generating the poor quality solution in GA, due to the random migration of VMs in the chromosome by selecting the random crossover point during crossover operation. But due to the random migration of VMs from one PM to other PM probability is more to search global optimal solution in GA.

In PSO, particles move their position due to the migration of the VMs from one PM to another PM by following the local and global best positions of the particle. This will lead to migration of the VMs placed on non-energy efficient PMs to the energy efficient PMs. Hence, PSO has fast convergence but due to the dependency on the local best particle position, sometimes the final solution falls in local optima. Therefore, before applying the crossover and mutation operations of GA, we need to apply PSO for improving the fitness value of the parent chromosomes, and thus obtaining the global optimum allocation of VMs to the PMs in less time unit.

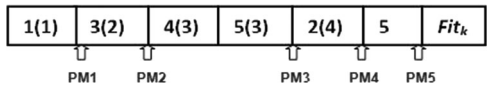


Fig. 4. Chromosome representation.

As shown in Fig. 3, Chr.s and Fit.s are the sth chromosome and its fitness value respectively. In the HGAPSO, finite number of chromosomes (Chr.1, Chr.2, ... ,Chr.s) are generated in the initial iteration and these chromosomes have fitness values (Fit.1, Fit.2, ... , Fit.s). After the generation of chromosomes by the initial iteration, let us sort the chromosomes in the decreasing order on the basis of their fitness value. Discard half of the chromosomes (less fittest) from the population and remaining half chromosomes (more fittest) will be considered for further operations. The best selected chromosomes are treated by the initial particles using PSO. Further, in the proposed HGAPSO, the quality of best selected chromosomes can be improved by using the PSO and then apply the GA operations (crossover and mutation operations respectively) to the said fine quality chromosomes. At the end of GA operations we need to select the fittest particles or chromosomes for the the next iteration. The HGAPSO algorithm terminates the iteration and thus allocates VMs to PMs when one of the termination conditions is satisfied. The proposed HGAPSO algorithm consist of three phases namely VMs Allocation (Phase 1), Task Scheduling (Phase 2), and VM Migration (Phase 3).

3.4.1 Phase 1: VMs Allocation

VMs Allocation phase consists of generation of chromosomes using GA, enhancement of chromosomes using PSO, and crossover and mutation operations using GA.

Chromosome Encoding. Fig. 4 shows the initial chromosomes and their corresponding fitness values. There are total 's' number of chromosomes which are generated by the initial population. Out of 's' chromosomes, '$s-1$' chromosomes are generated randomly and the remaining one chromosome is generated by using the FFD (Arrange VMs in decreasing order on the basis of their resource requirement, then allocate VMs to PMs using First-Fit) technique. The main reason behind this idea is to generate an initial population in such a manner that random generated chromosomes will give the diversity to the initial population, and FFD generated chromosomes will give an initial good solution.

Fitness Value. Fitness value of the chromosome 'k' is defined by Eq. (13) and its value is dependent on the euclidean distance (between 0 and 1)

$$fit_k = \frac{1}{\sum_{j=1}^{m} \sqrt{\left(\frac{pm_j}{pm_j^{max}}\right)^2 + (u_j^d - 1)^2}}. \quad (13)$$

Selection Operation. The selection of the best chromosome is done by the fitness value of the chromosome. If 's' is the size of a generation, then select half of the fittest chromosomes '$s/2$' for further computations and destroy the remaining half of the chromosomes. The selected chromosomes are thus considered for both the application of PSO, and GA operations (crossover and mutation). The probability of the chromosome considered for further computation is defined by Eq. (14). The chromosome with

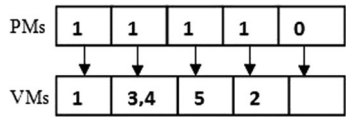


Fig. 5. Binary representation of a particle.

higher probability has more chances to go for further computations and the next generations

$$p_k = \frac{fit_k}{\sum_{k=1}^{s} fit_k}. \tag{14}$$

Particle Encoding and Position. After selecting the fittest chromosomes from the initial population, the mapping of chromosome to the particle is defined as follows: The position of ith particle is represented by an m-bit vector at iteration 't' is given as $X_i^t = \{x_1^t, x_2^t, \ldots, x_m^t\}$, where each bit of the position vector is either 0 or 1. In a chromosome those PMs contain VMs assigning 1 otherwise 0. The converted 'm' bit binary vector describes the position of the particle. Where 'm' is the number of PMs in the data center. The binary representation of a particle is shown in Fig. 5.

Particle Velocity. The ith particle velocity at iteration 't' is defined by 'n' bit velocity vector $V_i^t = \{v_1^t, v_2^t, \ldots, v_m^t\}$. This velocity vector changes the particle's position and thus changes the position of a VM from one PM to another PM. Each bit of the velocity is either 0 or 1. In the first iteration, we apply the random velocity to the particles. The random velocity is the binary stream of 0s and 1s of length 'm'.

Subtraction Operator. The subtraction operator is defined as the position difference between two VMs position vectors. In PSO, the subtraction operator is represented by symbol Θ. The resultant bit value of the subtraction of two position vectors $(X_i^t \; \Theta \; X_{i+1}^t)$ is 1 if both have same bit value; otherwise it is 0 (Ex: (1,1,1,0,1) Θ (1,0,1,0,0) = (1, 0, 1, 1,0)).

Addition Operator. The addition operator is defined as the velocity change of the particle. In PSO, it is represented by symbol $\oplus$ and is defined as $P_1V_1^t \oplus P_2V_2^t \oplus, \ldots, P_nV_n^t$. This represents the velocity change using V_1^t with probability P_1, and using V_n^t with probability P_n. In PSO, each probability $P_i, \sum_{i=1}^{n} P_i = 1$ is defined as an inertia coefficient. The results of the addition operation are explained as follows. For example, 0.3(1,1,0,1,0) $\oplus$ 0.7(1,1,1,0,0) = (1,1, $\neq$, $\neq$, 0). In this case, the probability of occurrence of 1 at 1st and 2nd bit positions is 1 and the probability of occurrence of 0 at the 5th bit position is 1. So the bit values at 1st, 2nd, and 5th positions are 1, 1, and 0 respectively, and the remaining positions are uncertain (denoted by $\neq$). To calculate the uncertain bit, we need three inertia weight coefficients P_{1i}, P_{2i}, P_{3i}, and these are defined as follows:

$$f(X_i^t) = \sum_{j=1}^{m} \sqrt{\left(\frac{P_j}{P_j^{max}}\right)^2 + (u_j^d - 1)^2} \tag{15}$$

$$P_{1i} = \frac{f(X_i^t)}{f(X_i^t) + f(X_{lbesti}^t) + f(X_{gbest}^t)} \tag{16}$$

$$P_{2i} = \frac{f(X_{lbesti}^t)}{f(X_i^t) + f(X_{lbesti}^t) + f(X_{gbest}^t)} \tag{17}$$

$$P_{3i} = \frac{f(X_{gbesti}^t)}{f(X_i^t) + f(X_{lbesti}^t) + f(X_{gbest}^t)}, \tag{18}$$

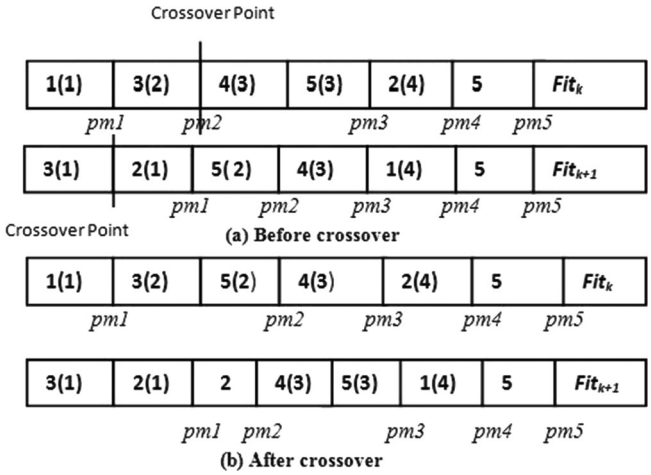


Fig. 6. Crossover operation.

where $f(X_i^t)$ denotes the fitness of a particle 'i' for the solution X_i^t; X_{lbesi}^t represents the local best position of particle i; X_{gbesi}^t represents the global best particle position. For calculation of inertia weight coefficients we need to consider high energy efficiency and maximum utilization with high probability

$$\begin{cases} uncertain & bit = q_1, \quad if \quad rand \le P_{1i} \\ uncertain & bit = q_2, \quad if \quad P_{1i} < rand \le P_{2i} \\ uncertain & bit = q_3, \quad if \quad P_{2i} < rand \le p_{3i}, \end{cases}$$

where q_1 is the respective bit of the particles velocity vector before updating; q_2 is the corresponding bit value of the local best particles vector; and q_3 is the corresponding bit of the best global particle velocity vector.

Multiplication Operator. The multiplication operator is represented by operator $\otimes$. It is defined for updating the particle position $(X_i^t \otimes V_i^{t+1})$ that represents the ith particle current position X_i^t and it is updated by velocity V_i^{t+1}. The computation rule for new particle position is as follows: (i) If the corresponding bit value of the velocity vector is 1, then the corresponding bit value of the new position vector is not adjusted; (ii) If the corresponding bit value of the velocity vector is 0, then it will be adjusted. For example (1,1,1,0,1) $\otimes$ (1,0,1,0,1), where (1,0,1,1,1) is the position vector and (1,0,1,0,1) is the velocity vector. The 2nd and 4th bits of the velocity vector are 0s and hence VMs are allocated to PMs when the 2nd and 4th positions of VMs are updated.

Finally, the PSO is used for computing the velocity and updated position as per the following details

$$V_i^{t+1} = P_1V_i^t \oplus P_2(X_{lbesti}^t \Theta X_i^t) \oplus P_3(X_{gbest}^t \Theta X_i^t) \tag{19}$$

$$X_i^{t+1} = X_i^t \otimes V_i^{t+1}. \tag{20}$$

Crossover Operation. After completion of PSO, one copy of the particles is stored temporarily and another copy will go for the crossover operation as shown in Fig. 6. The PSO generated particles are treated as chromosomes and the crossover operation is performed on the chromosomes. In our proposed work, we used a single point crossover operation since multi-point crossover operation migrates more number of VMs from one chromosome to another chromosome, and this not only degrades the fitness value of the

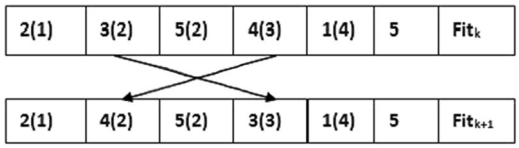


Fig. 7. Mutation operation.

chromosome but also creates more number of infeasible VMs. Thus for reallocating these infeasible VMs we need an extra computational time. In single point crossover operation we randomly select a map_i location and after this location, we exchange all the VMs between two chromosomes in the current generation.

Mutation Operation. After completion of the cross-over operation, we performed mutation operation on the chromosomes by randomly selecting two PMs in a chromosome. We select one VM on each of the selected PMs and then exchange the selected VMs. The details of mutation operation are shown in Fig. 7.

Reallocation of Infeasible VMs. In the case of PSO, when addition and subtraction operators are applied there is a chance of some particles reaching infeasible condition. The same infeasible condition arises when applying crossover and mutation operations on these chromosomes. Hence, to check the feasibility of all the particles and chromosomes, we need to use constraints (defined by Eqs. (7) to (10)) after applying each operation. Thus to select infeasible VMs from the infeasible particles or chromosomes, we need to remove infeasible VMs. Then, we need to reallocate the VMs to the PMs using FFD. Hence, by this way, the chromosomes or particles will not loose their fitness value. After completion of GA and PSO, we have one set of particles and another set of chromosomes. Using these sets of particles and chromosomes, we need to select the best particles or chromosomes using the roulette wheel selection operation as defined by Eq. (14) for the next generation.

Termination Condition. The computation process of the proposed HGAPSO (Algorithm 1) will be terminated if one of the following conditions is satisfied. (i) When the number of iterations is greater than the maximum set value of iterations. (ii) When there is no improvement in successive iterations or repetition of chromosomes and particle's value.

Algorithm 1 describes the allocation of VMs to PMs using the proposed HGAPSO. Where 't' is the iteration, t^{max} is the maximum number of iterations, and 's' is the generation size initialized by an integer number. Steps 1 to 11 generate the '$s-1$' number of random chromosomes. Step 12 describes the generation of one chromosome using FFD technique. Step 13 describes the termination condition of the proposed HGAPSO algorithm when specified termination condition such as ($t = t^{max}$ or repetition of same type of chromosomes and particles) is satisfied. Step 14 selects the '$s/2$' number of fittest chromosomes. Steps 15 to 16 describe the mapping of Chromosomes to Particles. Step 17 calls the Procedure 1 for performing the PSO operation. Step 18 calls the Crossover operation, and Step 19 calls the Procedure 2 for removing and refilling the infeasible VMs. Steps 20 to 22 describe the mutation operation and calls of Procedure 2 for each chromosome. Step 23 increments the iteration value.

Algorithm 1. Proposed HGAPSO

Initialize: t←1, t^{max}, s
Input: VMs list (VMs[]), PMs list (PMs[])
Output: Mapping of VMs to PMs
1: **for** i ← 1 to (s − 1) **do**
2: **while** $VMs[] \neq empty$ **do**
3: Randomly select vm_i from VMs[]
4: Randomly select pm_j from PMs[]
5: **for** Each d **do**
6: **if** $vm_i^d \leq pm_j^d$ **then**
7: Allocate vm_i to pm_j
8: Update $pm_j^d = pm_j^d - vm_i^d$
9: Remove vm_i form VMs[].
10: **else**
11: go to 4
12: Call FirstFit(Chr_s)
13: **while** Termination condition==false **do**
14: Select s/2 fittest Chromosomes.
15: **for** $i \leftarrow 1$ to s/2 **do**
16: $Part_i \leftarrow Chr_i$
17: Call ***Procedure*** *1*.
18: Call Crossover($Parent_1, Parent_2$)
19: Call ***Procedure*** *2*
20: **for** $i \leftarrow 1$ to $s/2$ **do**
21: Call Mutation(Chr_i)
22: Call ***Procedure*** *2*
23: $t \leftarrow t + 1$
24: Allocate VMs to PMs

Procedure 1. Applying PSO

1: **if** $t == 1$ **then**
2: $X_{gbest}^t \leftarrow null$
3: **for** i ← 1 to s/2 **do**
4: $X_i^t \leftarrow RandomPosition()$;
5: $V_i^t \leftarrow RandomVelocity()$;
6: $X_{lbesti}^t \leftarrow X_i^t$
7: **if** $f(X_{xlbesti}^t > f(X_{gbest}^t)$ **then**
8: $X_{gbest}^t \leftarrow X_{lbesti}^t$
9: **for** i ← 1 to s/2 **do**
10: Update V_i^{t+1} using Eq.19
11: Update X_i^{t+1} using Eq. 20
12: **if** $f(X_i^{t+1} > f(X_{lbesti}^t))$ **then**
13: $X_{lbest}^{t+1} \leftarrow X_i^{t+1}$
14: **if** $f(X_{lbesti}^{t+1} > f(X_{gbest}^t)$ **then**
15: $X_{gbest}^{t+1} \leftarrow X_{lbesti}^{t+1}$
16: ***Call Procedure 2***

Procedure 1 describes the PSO operations for enhancing the fitness value of chromosomes. Steps 1 to 5 describe the initialization of random position and random velocity of the particles during initial iteration. Steps 6 to 8 describe the assignment of currant particle position as a local best particle position (X_{lbesti}^t), and computation of a global best particle position (X_{gbest}^t). Updation of particle velocity V_i^{t+1} and particle position X_i^{t+1} are described by steps 9 to 11. Computation of both local and global best positions of particles are described by steps 12 to 15. Step 16 describes the calling of Procedure 2 for removing the infeasible VMs.

Procedure 2. Removing/Refilling of Infeasible VMs

1: **for** i $\leftarrow$ 1 to s/2 do **do**
2: $VM'[i] == empty$
3: **for** i $\leftarrow$ 1 to s/2 do **do**
4: **for** j $\leftarrow$ 1 to m do **do**
5: **for** k $\leftarrow$ 1 to d do **do**
6: **if** $\sum_{i=1}^{n} a_{ij} vm_i^d > pm_j^d$ **then**
7: Remove vm_i from pm_j
8: $VM'[i].add(vm_i)$
9: **for** i $\leftarrow$ 1 to s/2 **do**
10: Backfill VMs from $VM'[i]$ to Chr_i or $Part_i$ using FFD

Procedure 2 describes the removing of infeasible VMs and back-filling of removed infeasible VMs to the PMs. Initialization of empty VMs list for all individuals (chromosomes, particles) is described by steps 1 to 2. Removing of infeasible VMs (by checking of dimension of the PMs) and further adding of infeasible VMs are described by steps 3 to 8. Back-filling of infeasible VMs using FFD technique is described by step 10.

3.4.2 Phase 2: Task Scheduling

After allocation of user requested VMs to the PMs at the cloud data center, now VMs are ready for processing the user's generated applications. An application is divided into number of tasks of different lengths. Execution of tasks requires any kind of resources, such as CPU, RAM, Storage, or I/O devices. Hence, if a data center has 'k' number of users requested VMs allocated to the data center then kth user generated application A_k is represented as a stream of tasks like $A_k = \{t_0^k, t_1^k, t_2^k, .t_l^k\}$. Where t_l^k is the lth task of an application A_k. Like VMs and PMs, a task can also be defined as a vector unit. The dimension of the task vector is based on the type and amount of resources required to execute, such as CPU, RAM, Storage, etc. For the security and billing reason, the kth user generated application, A_k tasks are applied to their own purchased VMs. Further in the proposed HGAPSO, we consider all the tasks of CPU intensive. Therefore, before task scheduling task $t_i^k(mis)$ must satisfy the following constraints, and then this task will go to the respective VMs (VM, task).

$$\begin{cases} If \quad t_i^k(mis) \le vm_{small}^{mips} \quad \text{go to} \quad vm_{small} \\ ElseIf \quad vm_{small}^{mips} < t_i^k(mis) \le vm_{medium}^{mips} \text{ go to} \quad vm_{medium} \\ ElseIf \quad vm_{medium}^{mips} < t_i^k(mis) \le vm_{large}^{mips} \quad \text{go to} \quad vm_{large} \\ Else \quad vm_{x.large}^{mips} < t_i^k(mis) \le vm_{x.large}^{mips} \quad \text{go to} \quad vm_{x.large}. \end{cases}$$

If more than one task comes at the same time instance t which falls in the same conditional range such as $vm_{large} < t_i^k(mis), t_{i+1}^k(mis) \le vm_{x.large}^{mips}$, then one task will wait in the queue of $vm_{x.large}$ till previously assigned task is not completed its execution. Hence, in our proposed work, we consider the sequential task dependency in an application A_k such as task t_{i+1}^k, which will be executed after the execution of task t_i^k.

The Service Level Agreement (SLA) is a contractual document between the customer and the cloud service provider. The SLA violation in the cloud computing depends on the type and amount of infrastructure purchased by the customer and the quality of service provided by the service provider. If there is a quality mismatch from the contractual document, then service provider will pay the amount of money to the customer based on their contractual document. If the application running time is longer, then in that case, in place of defined deadline for each task, the average response time or total execution time of the application does matter. Hence user defines the application deadline in terms of execution time, and this execution time depends on the amount of resources in terms of VMs purchased by the user. Hence, on the basis of purchased VMs configuration, the customer thus defines the deadline time of each application. Then the formulation of SLA violation is defined as follows:

Let us consider a customer purchased different type of VMs in different quantity such as $vm_{small}(n1)$, $vm_{medium}(n2)$, $vm_{large}(n3)$, $vm_{x.large}(n4)$ and application A_k is running on these purchased VMs then deadline D_k for the application A_k is defined as follows:

$$D_k = \frac{\sum_{k=1}^{l} t_k^{mis}}{\sum_{i=1}^{n1+n2+n3+n4} vm_i^{mips}}. \tag{21}$$

If waiting time $(w_i^k(t))$, and execution time $(ex_i^k(t))$ of a task t_i^k for application A_k, are considered then the response time $r_i^k(t) = w_i^k(t) + ex_i^k(t)$. Therefore the response time of an application A_k $(R_k(t))$ having 'l' number of tasks is defined as

$$R_k(t) = \sum_{i=1}^{l} w_i^k(t) + r_i^k(t). \tag{22}$$

Hence, SLA violation condition of an application A_k is defined as

$$\begin{cases} No \quad SLA \quad violation \quad if \quad (R_k(t) \le D_k(t)) \\ SLA \quad violation \quad otherwise. \end{cases}$$

The amount of SLA violation of given application A_k is dependent on the time difference between the response time $(R_k(t))$ and the deadline time $(D_k(t))$, such as $(R_k - D_k)$. Since the commercial cloud service provider such as Amazon provides EC_2 VM instances on the rent basis by creating the different types of user requested VMs to the data center during specified period of time. Hence, each type of VMs such as small, medium, large, and x.large, have different amount of rent such as \$0.0065 per Hour for small VM instance [31]. Hence, we consider α, β, γ, and δ are the rents of small, medium, large, and x.large VM types for unit time duration respectively. Further, if SLA violation occurs for a user then the cloud service provider will pay the penalty to the user, and this penalty is defined in the case of different type of VMs instances given by

$$\begin{cases} vm_{small}(SLA) = (R_k(t) - D_k(t)) * \alpha \\ vm_{medium}(SLA) = (R_k(t) - D_i(t)) * \beta \\ vm_{large}(SLA) = (R_k(t) - D_k(t)) * \gamma \\ vm_{x.large}(SLA) = (R_k(t) - D_k(t)) * \delta, \end{cases}$$

where $vm_{small}(SLA)$, $vm_{medium}(SLA)$, $vm_{large}(SLA)$, $vm_{x.large}(SLA)$ are the penalty amounts to be paid by the

service provider to the customer in the case of small, medium, large, and x.large VMs instances respectively.

3.4.3 Phase 3: VMs Migration

The allocated VMs in the PMs will be gradually terminated from the system as their related tasks are completed. The termination of VMs from the PMs give the opportunity to re-optimize the allocation of VMs to PMs by migrating the VMs from the underutilized PMs to the energy efficient PMs. Hence by VM migration, we consolidate the VMs into less number of energy efficient PMs and shutdown the ideal PMs; thus more energy saving at the cloud data center. The objective function for optimum VM migration is to migrate the VMs from the source node to the destination node so that filling the maximum number of VMs at the destination node within their capacity, and switched-off source node after VMs migration. Hence by this way we can maximize the number of ideal PMs to be turned off at the data center.

Let us consider a set of non ideal PMs 's', $|s| < m$. The objective function for the optimal VM migration and consolidation is defined by

$$max \quad y = \sum_{i \in s} P_i^{min} y_i - \sum_{i \in s} \sum_{j \in s} \sum_{k=1}^{n_i} P'_K z_{ijk}, \tag{23}$$

where y_i = 1 if pm_i used for the VM allocation, otherwise $y_i = 0$; Variable z_{ijk} = 1 if vm_k migrate from pm_i to pm_j otherwise it is 0; P'_K is the power consumption of vm_k; n_i is the set of VMs to be migrate from pm_i. The objective function in Eq. (23) should satisfy the following migration constraints during the VMs migration

$$\sum_{i \in s} \sum_{k=1}^{n_i} vm_k^d z_{ijk} \leq pm_j^d (1 - y_i) \forall d \in \{mips, ram, storage\} \tag{24}$$

$$\sum_{j \in s} \sum_{k=1}^{n_i} z_{ijk} = n_i y_i \quad \forall i \in s, i \neq j. \tag{25}$$

The constraints satisfaction of different resources is defined in Eq. (24). Migration of a set of VMs (n_i) from pm_i to pm_j is defined in Eq. (25). The proposed migration policy consists of two stages for calculating when and where to migrate VMs at the data center. Further to avoid SLA violation and minimize energy consumption at the data center, we set the lower threshold value of CPU utilization (u_l) for each PM at the data center. In the first stage, we decide when to migrate the VMs (selection of source node) and in the second stage, we decide where to migrate the VMs (selection of destination node). The migration of VMs from pm_i will take place if the current CPU utilization of pm_i is less than the lower threshold value u_l such as $(u_j^{cpu} < u_l)$, and the cost of migration for all the VMs allocated to pm_i should be less than the rent of VMs as decided by cloud services provider during the left over time t_l. Hence the selection of a set of VMs n_i from the source node pm_i is defined by

$$n_i = \begin{cases} migration \quad of \quad n_i \quad if \quad u_i^{cpu} < u_l \quad and \\ c_{mig}^t * n_i \; < \; n_i r * t_l, \quad r \in \{\alpha, \beta, \gamma, \delta\}, \quad i \in s \\ No \quad migration \quad otherwise. \end{cases} \tag{26}$$

The left over time (t_l) is the difference between total life time (t_t) of vm_i and currant time (t_c) of vm_i such as $t_l = t_t - t_c$. The total migration cost (c_{mig}^t) for the set of VMs n_i is depends on two factors such as SLA penalty during shut down time s_t and migration overhead cost c_{mig}^o. Therefore total migration cost is defined by

$$c_{mig}^t = \sum_{k=1}^{n_i} s_t * r + \sum_{k=1}^{n_i} c_{mig}^o. \tag{27}$$

The amount of SLA penalty during the VM shutdown time is equal to the rent of VM during that period of time paid by the user to the service provider. Further the migration overhead cost c_{mig}^o is dependent on the network bandwidth, size of memory content to be copied from the source node to the destination node etc. Hence the fixed power consumption P'_k is assigned to each VM instance on the basis of their type such as small, medium, large, and x.large as a migration overhead cost. In the second stage, we arrange all the VMs collected by the sources nodes in the decreasing order on the basis of their resource requirement and then allocate the VMs to the PMs using first fit decreasing technique.

3.5 Branch-and-Bound Based Exact Algorithm

To check the performance of proposed HGAPSO algorithm with benchmark solution, we designed an exact algorithm which is search tree based branch-and-bound method in the form of multi-objective VMs allocation at the data center. The exact algorithm gives the global optimal allocation of VMs to PMs at the data center. Hence we compared our proposed HGAPSO algorithm with this exact algorithm, and the details of exact algorithm are as follows. Let 'N' be the node at level 'l' of the search tree which is associated with a lower bound $lb(N)$, an upper bound $ub(N)$ with the following data:

$S'(N)$: subset of unallocated VMs $(|S'(N)| = n - l)$; $v_k(N)$: number of PMs of type k that have have been initialized; $v(N)$: total number of PMs that have been initialized $(v(N) = \sum_{k=1}^{m} v_k(N))$; $p(N)$: total power consumption of the PMs have been initialized $(p(N) = \sum_{k=1}^{m} v_k(N) p_k)$.

The branching strategy implemented is similar to that of Martello and Toth [32] for the bin packing problem. The root node N_0 in the search tree corresponds to the empty solution and each node at level $l \geq 1$ corresponds to the partial solution. A child node is created by allocating the largest VM to an initialized PM or to an empty PM of type k. Hence a node N in the search tree may have maximum $m + v(N)$ child nodes. Further a given node N, a child node N^+ corresponding to allocating vm_i into a PM of type k is created only if the following conditions hold good:

i) (C_1) $v(N) \leq v^u$ (v^u is an upper bound on the total number of PMs are used in an optimal solution);
ii) (C_2) $v_k(N) < v_k^u$ (v_k^u is an upper bound on the number of PMs of type k that are loaded in an optimal solution);
iii) (C_3) $c(N) + c_K < ub(N)$;
iv) (C_4) $d_i \leq D_k$, $\forall d \in \{mips, ram, storage\}$ (or $d_i \leq D_k$ if k is initialized PM.

Hence if the above conditions are met, then the lower bound $lb(N^+)$ is computed. Further if $lb(N^+) \geq ub(N)$ then the child node is pruned.

TABLE 3
PM Configuration

PM Type	PE	MIPS	RAM	STORAGE
ProLiantM110G5XEON3075	2	2,660	4	160
IBMX3250Xeonx3480	4	3,067	8	250
IBM3550Xeonx5675	12	3,067	16	500

4 Performance Evaluation

4.1 Experimental Setup

We used Java language for the implementation of the proposed HGAPSO and VM migration policy. Further we compared our proposed HGAPSO algorithm with bench mark solution provided by the branch-and-bound based exact algorithm. In our experiment, we consider three different types of PMs and four different types of VMs for the creation of heterogeneous data center environment in the form of VMs and PMs. The PMs power consumption and other resources characterises of PMs are downloaded from the IBM [33] and Dell [34] websites. Further, the Amazon based VM instances used for the experiment are downloaded from Amazon website [31]. The configurations of PMs and VMs used for the simulation are shown in Tables 3 and 4 respectively. Further, for all VMs (PMs) combinations, we consider same number of VMs and PMs (all types). For example, 100 VMs and 60 PMs combination of data center consists of 25 small, 25 medium, 25 large and 25 x.large type of VMs; 20 Proliant, 20 IBM3250, and 20 IBM3550 type of PMs in heterogeneous environment. To check the performance of proposed HGAPSO based VMs allocation algorithm in homogeneous environment, we considered only single type PM (Type 2 PM) in the data center. The detailed configuration of Type 2 PM is given in Table 3. Further to avoid congestion control and failure of PM due to excess heat generation, we kept a buffer region for each type of resource. Hence in our experiment, we set the maximum limit of CPU, RAM, and Storage utilization are (70, 80, and 95 percent) respectively.

4.2 Results

For evaluating the performance of the proposed HGAPSO algorithm as compared to the benchmark solution in terms of energy consumption and resources utilization, we considered different combinations of VMs and PMs in two different data center environments (Homogeneous and Heterogeneous). The number and types of PMs used for different amount of VMs allocation in heterogeneous and homogeneous data center environment is shown in Table 5. The proposed HGAPSO algorithm utilizes near equal number of PMs as compared to the exact algorithm. Since our proposed hybrid approach has improved the fitness of the chromosomes by applying PSO before applying the crossover and mutation operations. Thus the crossover and mutation operations produced good quality children, and thereby achieving the near global optimum solution for VMs allocation. Thus results in less number of PMs used for the VMs allocation which is nearly equal to the global optimum solution given by the exact algorithm.

TABLE 4
VM Configuration

VM Type	PE	MIPS	RAM	STORAGE
Small	1	500	0.5	40
Medium	2	1,000	1	60
Large	3	1,500	2	80
X.large	4	2,000	3	100

TABLE 5
PMs Used for VMs Allocation at Data Center

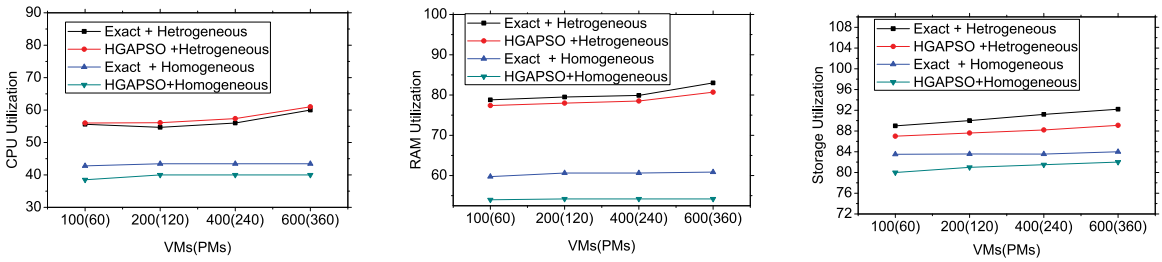
VMs(PMs)	HGAPSO		Exact Algorithm	
	Heterogeneous	Homogeneous	Heterogeneous	Homogeneous
100(60)	(8,19,3)	37	(7,17,5)	34
200(120)	(15,31,12)	71	(13,28,11)	67
400(240)	(33,71,24)	139	(29,68,14)	134
600(360)	(45,99,55)	205	(41,94,35)	195

Figs. 8a, 8b, and 8c show the CPU, RAM, and Storage utilization respectively of the data center in heterogeneous and homogeneous environment for different combinations of VMs and PMs using HGAPSO, and exact algorithm. The CPU utilization, RAM utilization, and Storage utilization of the proposed HGAPSO are slightly low as compared to the bench mark algorithm on each VMs (PMs) combinations due to the less number of energy efficient PMs used for the VMs allocation. Since more number of PMs are switched-off and thus minimizes the resources wastage. The resources utilization line is almost straight in homogeneous data center environment for both VMs allocation techniques because of used PMs are increasing in the same proportion as number of VMs are increasing resulting in less variation in resources utilization.

Figs. 9a and 9b show the power consumption of the data center for different VMs (PMs) combinations using HGAPSO and exact algorithm in heterogeneous and homogeneous data center environment respectively. The power consumption of the data center in the case of HGAPSO algorithm is slightly high (< 10 percent) when compared to the benchmark solution. Since combination of type1, type2, and type3 PMs used by the proposed HGAPSO is almost equal as compared to that of exact algorithm; hence the power consumption in the case of HGAPSO is slightly high when compared to that of exact algorithm.

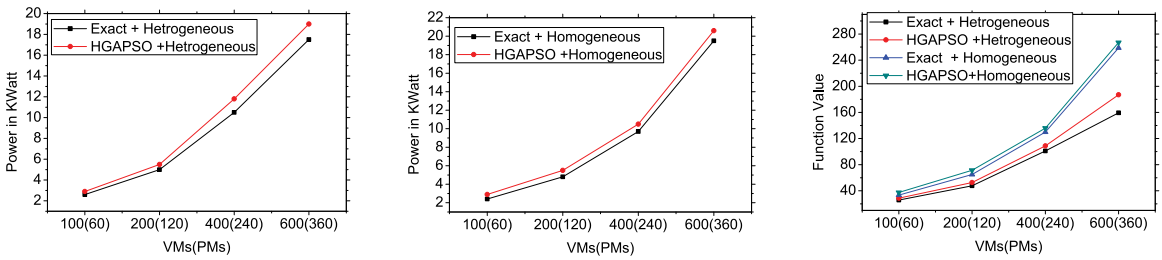
Fig. 9c shows the objective function value based on resources utilization and power consumption for two different VMs allocation techniques over different combinations of VMs (PMs) in homogeneous and heterogeneous data center environment. Since the function value 'f' is dependent on the power consumption and resources utilization. The power consumption and resource utilization in the case of HGAPSO is reaching near to the global optimal solution provided by the exact algorithm. Thus the overall function value of the proposed HGAPSO is slightly high when compared to that of exact VM allocation algorithm in both heterogeneous and homogeneous environments.

After allocation of VMs to the PMs, we applied the random tasks on regular time interval (1 second) for the calculation of energy consumed by the data center during specified duration of time. Hence in our simulation, we generated four different tasks on every 1 second during the period of 200 seconds. The task scheduling on VMs is done



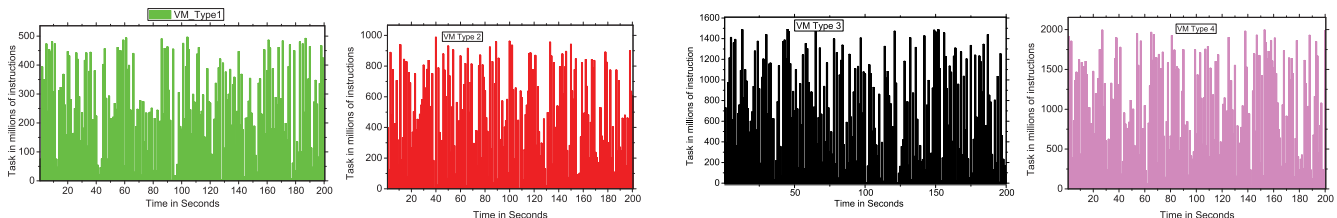
(a) CPU Utilization (b) RAM Utilization (c) Storage Utilization

Fig. 8. CPU, RAM, and storage utilization in heterogeneous and homogeneous data center.



(a) Power Consumption (b) Power Consumption (c) Function Value

Fig. 9. Power consumption and function value in heterogeneous and homogeneous data center.



(a) Type VM1 (b) Type VM2 (c) Type VM3 (d) Type VM4

Fig. 10. Task allocation to different VMs.

by using the VMs (task) conditions specified in phase 2 (task scheduling). The random allocated tasks to Type1 VMs, Type2 VMs, Type3 VMs, and Type4 VMs during 200 seconds are shown in Figs. 10a, 10b, 10c, and 10d respectively.

We considered the processor intensive tasks in millions of instructions throughout our simulation. The amount of energy consumed by the data center using different VMs allocation without VMs migration policy when 600 VMs and 360 PMs combination used at the data center for both heterogeneous and homogeneous environments is shown in Fig. 11. The overall energy consumption is approximately (< 10 percent) high in the case of HGAPSO when compared to that of benchmark solution in both environments. This is due to the near global optimal placement of VMs to PMs by our proposed HGAPSO algorithm and resulting in switching-off more number of ideal PMs at the data center. The global optimal placement of VMs allocated to the energy efficient PMs will further lead to less energy consumption at the data center.

To check the performance of our proposed VM migration policy, we generated the user requested VMs in discrete time interval at the data center. In this experiment we considered 600 PMs at the data center (e.g., 200 type1, 200 type2, 200 type3). Further to derive the system model, each user has to send the request to the service provider and thereby getting the required VMs and the type of instances (small, medium, large, x.large). The range of VMs requested by the user is in between [1,100]. The life time of the VMs is uniform in the the range of [30 to 200 s]. To migrate a VM from source node to the destination node, we used constant migration overhead cost c^{o}_{mig} in terms of power consumption for each type of VMs such as 10 watt (small), 20 watt (medium), 30 watt (large), and 40 watt (x.large). On the arrival of each new user's request at the data center, we applied our proposed HGAPSO algorithm for VM allocation. After completion of task by the VM, we terminate the VM from the PM, and further applied the proposed VM migration policy on the PMs which are under utilized. In this experiment we consider the lower utilization threshold of ($u_l = 50$ percent) for all the PMs at the data center.

Fig. 12. shows the combined performance of proposed HGAPSO with VM migration policy known as

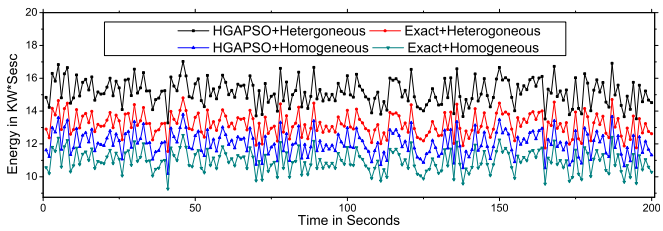


Fig. 11. Energy consumption without VM migration .

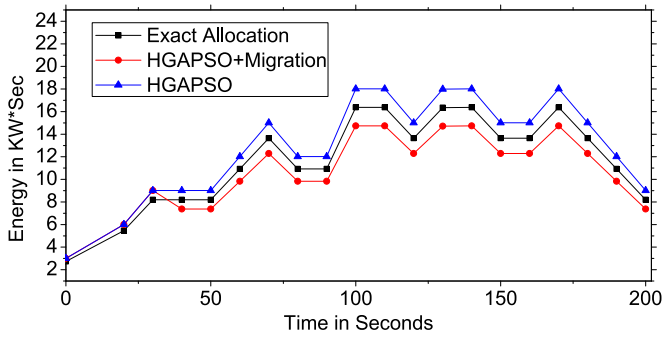


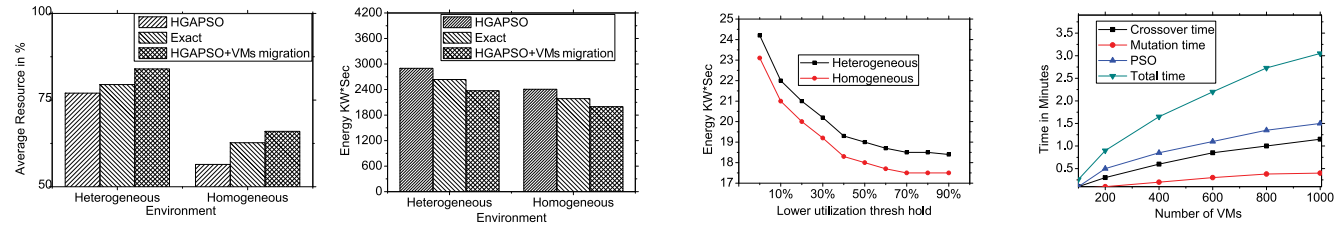
Fig. 12. Energy consumption with VM migration.

(HGAPSO+VM Migration) in comparison with both HGAPSO and exact algorithms. The proposed HGAPSO +VM Migration saves 20 and 10 percent energy at the data center when compared to HGAPSO and exact algorithms respectively. The combined approach gives better results in terms of energy saving over exact algorithm due to switching-off underutilized PMs at the data center using VM migration technique. Hence by this way, we can increase the effective CPU utilization of PMs at the data center and consolidate the VMs to less number of energy efficient PMs at the data center. The migration of VMs at the data center is based on regular time interval. In our experiment, we used the migration time interval of 30 seconds. After each 30 seconds, we gathered the utilization status of each used PMs at the data center and made a list of VMs which are allocated to under utilized PMs. Further we migrate all the VMs in the list using migration policy as defined in Phase 3 and then we can switch-off underutilized PMs at the data center.

Figs. 13a and 13b show the average resource utilization and overall energy consumption using 600 VMs and 360 PMs in homogeneous and heterogeneous data center environments respectively. The average resource utilization (Fig. 13a) for our proposed HGAPSO+VM Migration algorithm is 12 and 14 percent more when compared to that of exact algorithm in heterogeneous and homogeneous environments respectively. Further the overall energy consumption is (9.8 percent) less as compared to that of exact algorithm in both heterogeneous and homogeneous environments.

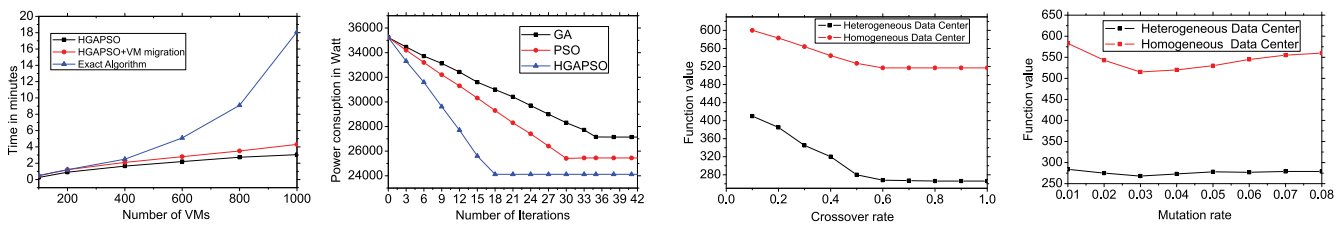
To check the performance of the proposed VM migration policy, in this experiment we set different values of lower threshold utilization (u_l) in between [0 to 90 percent]. Fig. 13c shows the energy consumption of data center in both environments while keeping different values of u_l. In the case of $u_l = 0$ percent, there is no VM migration and this results in high energy consumption but there is no SLA violation. While increasing u_l value we observe that more number of VMs are migrated from under utilized PMs to the energy efficient PMs in the data center. This results in lower energy consumption at the data center but there is a SLA violation. Hence we an optimal point for energy saving with less SLA violation. Thus in our experiment, we consider lower threshold utilization ($u_l = 50$ percent) and thereby selecting the optimal point for not only saving the energy of the data center but also to reduce the SLA violation.

To check the scalability of the proposed algorithm, we conducted the experiment on HP Compaq LE1902x with 8 GB RAM, 3.40 GHz i-7 Processor and calculated the execution time of the proposed HGAPSO algorithm. The execution time of proposed HGAPSO algorithm is mainly dependant on the crossover operation, mutation operation, and PSO. Hence total execution time of proposed HGAPSO algorithm is shown in Fig. 13d. The total execution time for the placement of 1,000 VMs to the data center is approximately 3.5 minutes which is scalable in nature for the large data center. Further the execution time of the combined HGAPSO+VM migration algorithm is shown in Fig. 14a. The execution time of exact algorithm is going exponentially high when more number of VMs (PMs) combination is used at the data center, where as the execution time of the proposed HGAPSO+VM migration algorithm is linearly increasing when more number of VMs (PMs) combinations is used at the data center. Hence for the large data center, the proposed HGAPSO+VM migration algorithm not only saves the energy consumption but also avoids the SLA violation in terms of user response time.



(a) Average Resource Utilization (b) Energy Consumption (c) Energy reduction on threshold (d) Execution Time of HGAPSO

Fig. 13. Average resources utilization, energy consumption, energy reduction on threshold, execution time of HGAPSO.



(a) HGAPSO+VM Migration (b) Convergence (c) Crossover Rate (d) VMs Mutation Rate

Fig. 14. Execution time of HGAPSO+VM migration, convergence, crossover rate, mutation rate.

TABLE 6
HGAPSO Parameters

Parameters	GA	PSO	HGAPSO
Population size	10	10	10
Max. Iteration	50	50	50
Crossover rate	0.6	-	0.6
Mutation rate	0.03	-	0.03
Selection	Roulette Wheel	-	Roulette Wheel
(c1, c2,w)	-	(0.5, 0.5,0.5)	(0.5, 0.5,0.5)

The minimization of power consumption on each iteration is considered for three evolutionary algorithms such as GA, PSO, and HGAPSO and its convergence is shown in Fig. 14b. The convergence of HGAPSO is fast as compared to that of GA and PSO since we improved the fitness of the chromosomes by applying the PSO, just before using the crossover and mutations operations. Further this will result in less number of iterations required for the convergence of our proposed HGAPSO algorithm. For checking the reliability of the proposed HGAPSO algorithm, we conducted algorithm 15 times on each VMs (PMs) combinations and calculated the frequency of the same solution. The final allocation of VMs to PMs at the data center is based on the solution which has the highest frequency. Hence by this way, we calculated the confidence interval $(> 90 \text{ percent})$ of the highest frequent solution for the HGAPSO.

The generation size (S), number of generations (t), crossover rate (CR), mutation rate (MR) are the important parameters for the performance of proposed HGAPSO algorithm. Fig. 14c shows the performance of the proposed HGAPSO over different crossover rates. When the crossover rate is low then the objective function value is high; thus the proposed HGAPSO converged to non-global optimal point and takes more number of iterations. Further, by increasing the crossover rate, the objective function value is continuously going down, and an optimal crossover point of 0.6 is considered for the proposed HGAPSO. Further increasing the crossover rate from 0.6 increases the objective function value due to the early stagnation of the chromosomes resulting in sub-optimal value of objective function.

The performance of HGAPSO on mutation rate is shown in Fig. 14d. The mutation rate of 0.03 gives the near global optimum allocation of VMs to the PMs by the proposed HGAPSO. The value of mutation rate less then 0.03 gives very little divergence in the solution space resulting in sub-optimal solution. Mutation rate greater then 0.03 gives higher diversity in the solution thus HGAPSO diverse away from the final near global optimum solution. To check the performance of proposed HGAPSO on solution size, we set different generation sizes (5 to 25). The performance of HGAPSO is high for generation size 10. The solution size less than 10 takes more number of iterations and gives sub-optimal solution due to less diversity in the solution space. The solution size more then 10 gives the same solution as that of 10 but the simulation running time is increased due to high of computation time of the proposed HGAPSO.

The best values of the parameters for our proposed HGAPSO algorithm are given in Table 6. The best values of crossover and mutation parameters are calculated by checking the performance of HGAPSO on each crossover point between 0.1 and 1 through increasing the crossover rate from 0.01 to 0.08. In this process, we kept all other parameters such as population size, constants ($c1$, $c2$) and weight w are constant. After getting best values of crossover and mutation operations increase the population size and fixed the minimum generation size which gives optimal allocation of VMs to PMs. The best constant values $c1$, $c2$ and weight value w are calculated by taking initial values $c1$ = 0.1, $c2$ = 0.1, and w = 0.1 and increase these values by 0.1 till it will not reach to 0.9. Let us consider the minimum values of $c1$, $c2$, and w so that there will not be any further performance improvement in the proposed HGAPSO.

4.3 Time Complexity Analysis

The time complexity of proposed HGAPSO with VM migration technique is based on GA, and PSO. Let us consider maximum iteration size is 'k', individual size is 'm' (number of PMs), Number of VMs is 'n', generation size is 's', and crossover point for each generation is 'p'. The time complexity of GA is dependent on the individual generation, crossover, and mutation operations. The time complexity of PSO is dependent on (Particle generation, Initial velocity setting, Position calculation, global fitness calculation, local fitness calculation, updating in velocity, position updating, back fill of missed VMs, and removing of duplicate VMs for specified iterations). Further the First Fit Decreasing (FFD) based VM migration requires O($nlogn$) time complexity. HGAPSO = O(O(Individual generation) + Total iterations $*$ (O(Fitness calculation $*$ Size of generation)+ O(Crossover $*$ Size of generation) + O(Mutation $*$ Size of generation)) + O (PSO) + O(VM migration) Hence resultant time complexity of HGAPSO = O(O($n * s$) + O(k $*$ (O($m * s$) + O($n* m * s - p * m * s$) + O(s))) + O(k $*$ (O($s * m$) + O($s * m$) + O(s) + O(s) + O($n * m * s$)) + O($nlogn$)) which is equal to O($n. * m * s * k$) + $nlogn$ polynomial in nature.

5 Conclusion and Future Work

This paper highlights the proposed hybrid approach for multi-objective function VMs allocation and VMs migration at the data center in both homogeneous and heterogeneous environments up to 600 VMs and 360 PMs. Due to the space limitations, we considered four types of VMs but the proposed algorithm can be extended for more number of VMs (PMs) combinations. The combined approach for VM allocation and VM migration known as (HGAPSO+VM migration) not only saves the energy consumption by 9.8 percent less but also improves the average resource utilization by 13 percent more in homogeneous and heterogeneous environments when compared to bench mark solution achieved by exact algorithm.

References

[1] M. Uddin and A. A. Rahman, "Energy efficiency and low carbon enabler green it framework for data centers considering green metrics," *Renewable Sustain. Energy Rev.*, vol. 16, no. 6, pp. 4078–4094, 2012.

[2] M. Pedram, "Energy-efficient datacenters," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 31, no. 10, pp. 1465–1484, Oct. 2012.

[3] H. Hajj, W. El-Hajj, M. Dabbagh, and T. R. Arabi, "An algorithm-centric energy-aware design methodology," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 22, no. 11, pp. 2431–2435, Nov. 2014.

[4] N. Kim, J. Cho, and E. Seo, "Energy-credit scheduler: An energy-aware virtual machine scheduler for cloud systems," *Future Generation Comput. Syst.*, vol. 32, pp. 128–137, 2014.
[5] D. Whitley, V. S. Gordon, and K. Mathias, "Lamarckian evolution, the Baldwin effect and function optimization," in *Proc. 3rd Parallel Problem Solving from Nature*, 1994, pp. 5–15.
[6] Y. Gao, H. Guan, Z. Qi, Y. Hou, and L. Liu, "A multi-objective ant colony system algorithm for virtual machine placement in cloud computing," *J. Comput. Syst. Sci.*, vol. 79, no. 8, pp. 1230–1242, 2013.
[7] D. W. Corne, N. R. Jerram, J. D. Knowles, M. J. Oates, and J. Martin, "PESA-II: Region-based selection in evolutionary multi-objective optimization," in *Proc. Genetic Evol. Comput. Conf.*, 2001, pp. 283–290.
[8] H. Ishibuchi, T. Yoshida, and T. Murata, "Balance between genetic search and local search in memetic algorithms for multi-objective permutation flowshop scheduling," *IEEE Trans. Evol. Comput.*, vol. 7, no. 2, pp. 204–223, Apr. 2003.
[9] R. Ge, X. Feng, and K. W. Cameron, "Performance-constrained distributed DVS scheduling for scientific applications on power-aware clusters," in *Proc. ACM/IEEE Conf. Supercomputing*, 2005, p. 34.
[10] C.-H. Hsu and W.-C. Feng, "A power-aware run-time system for high-performance computing," in *Proc ACM/IEEE Conf. Supercomput.*, 2005, p. 1.
[11] B. Kveton, P. Gandhi, G. Theocharous, S. Mannor, B. Rosario, and N. Shah, "Adaptive timeout policies for fast fine-grained power management," in *Proc. Nat. Conf. Artificial Intell.*, 2007, vol. 22, no. 2, pp. 1795–1800.
[12] C.-M. Wu, R.-S. Chang, and H.-Y. Chan, "A green energy-efficient scheduling algorithm using the DVFS technique for cloud datacenters," *Future Generation Comput. Syst.*, vol. 37, pp. 141–147, 2014.
[13] Y. Ajiro and A. Tanaka, "Improving packing algorithms for server consolidation," in *Proc. Int. Comput. Meas. Group Conf.*, 2007, pp. 399–406. [Online]. Available: http://dblp.uni-trier.de/db/conf/cmg/cmg2007.html#AjiroT07
[14] E. G. Coffman Jr., M. R. Garey, and D. S. Johnson, "Approximation algorithms for bin packing: A survey," in *Approximation Algorithms for NP-Hard Problems*, D. S. Hochbaum, Ed. Boston, MA, USA: PWS Publishing Co., 1997, pp. 46–93. [Online]. Available: http://dl.acm.org/citation.cfm?id=241938.241940
[15] A. Beloglazov and 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.*, vol. 24, no. 7, pp. 1366–1379, Jul. 2013.
[16] A.-P. Xiong and C.-X. Xu, "Energy efficient multiresource allocation of virtual machine based on PSO in cloud data center," *Math. Problems Eng.*, vol. 2014, pp. 1–8, 2014.
[17] Y. Gao, H. Guan, Z. Qi, Y. Hou, and L. Liu, "A multi-objective ant colony system algorithm for virtual machine placement in cloud computing," *J. Comput. Syst. Sci.*, vol. 79, no. 8, pp. 1230–1242, 2013.
[18] S. Wang, Z. Liu, Z. Zheng, Q. Sun, and F. Yang, "Particle swarm optimization for energy-aware virtual machine placement optimization in virtualized data centers," in *Proc. IEEE Int. Conf. Parallel Distrib. Syst.*, 2013, pp. 102–109.
[19] J. Xu and J. A. Fortes, "Multi-objective virtual machine placement in virtualized data center environments," in *Proc. IEEE/ACM Int. Conf. Green Comput. Commun. Int. Conf. Cyber Physical Soc. Comput.*, 2010, pp. 179–188.
[20] M. Tighe and M. Bauer, "Integrating cloud application autoscaling with dynamic VM allocation," in *Proc. Netw. Oper. Manage. Symp.*, 2014, pp. 1–9.
[21] J. J. Prevost, K. Nagothu, B. Kelley, and M. Jamshidi, "Prediction of cloud data center networks loads using stochastic and neural models," in *Proc. 6th Int. Conf. Syst. Syst. Eng.*, 2011, pp. 276–281.
[22] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes, "AGILE: Elastic distributed resource scaling for infrastructure-as-a-service," in *Proc. USENIX Int. Conf. Automated Comput.*, 2013, pp. 69–82.
[23] Z. Xiao, W. Song, and Q. Chen, "Dynamic resource allocation using virtual machines for cloud computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1107–1117, Jun. 2013.
[24] S. K. Garg, C. S. Yeo, A. Anandasivam, and R. Buyya, "Environment-conscious scheduling of HPC applications on distributed cloud-oriented data centers," *J. Parallel Distrib. Comput.*, vol. 71, no. 6, pp. 732–749, 2011.
[25] M. Shojafar, N. Cordeschi, D. Amendola, and E. Baccarelli, "Energy-saving adaptive computing and traffic engineering for real-time-service data centers," in *Proc. IEEE Int. Conf. Commun. Workshop*, 2015, pp. 1800–1806.
[26] T. V. T. Duy, Y. Sato, and Y. Inoguchi, "Performance evaluation of a green scheduling algorithm for energy savings in cloud computing," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. Workshops Phd Forum*, 2010, pp. 1–8.
[27] C. Ghribi, M. Hadji, and D. Zeghlache, "Energy efficient VM scheduling for cloud data centers: Exact allocation and migration algorithms," in *Proc. 13th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, 2013, pp. 671–678.
[28] A. Karthick, E. Ramaraj, and R. G. Subramanian, "An efficient multi queue job scheduling for cloud computing," in *Proc. World Congr. Comput. Commun. Technol.*, 2014, pp. 164–166.
[29] D. M. Quan, F. Mezza, D. Sannenli, and R. Giafreda, "T-Alloc: A practical energy efficient resource allocation algorithm for traditional data centers," *Future Generation Comput. Syst.*, vol. 28, no. 5, pp. 791–800, 2012.
[30] L. Minas and B. Ellison, *Energy Efficiency for Information Technology: How to Reduce Power Consumption in Servers and Data Centers*. Santa Clara, CA, USA: Intel Press, 2009.
[31] Amazon "EC2 Instances," 1999. [Online]. Available: https://aws.amazon.com/ec2/instance-types/
[32] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. New York, NY, USA: Wiley, 1990.
[33] IBM, "Power Model," 1999. [Online]. Available: http://www.dell.com/
[34] Dell, "Power Model," 1999. [Online]. Available: http://www-03.ibm.com/systems/power/hardware/

Neeraj Kumar Sharma received the BE degree from JIT, Khargone, Madhya Pradesh, India, in 2005, and the ME degree from IET, Indore, Madhya Pradesh, India, in 2011. He is currently working toward the PhD degree at NIT Karnataka, Surathkal, Mangalore, India. His current research is on energy efficient resources management in cloud data center. He has four IEEE conference publications so far.

G. Ram Mohana Reddy received the BTech degree from S.V. University, Tirupati, Andhra Pradesh, India, in 1987, the MTech degree from Indian Institute of Technology, Khargpur, India, in 1993, and the PhD degree from The University of Edinburgh, United Kingdom, in 2005. Currently he is a professor and head of the Department of Information Technology, National Institute of Technology Karnataka, Surathkal, Mangalore, India. His research interests include affective computing, bio-inspired cloud/green computing, cognitive health care analysis, and social multimedia/social network analysis. He has more than 160 research publications in reputed international journals, conference proceedings and books/book chapters. He is a senior member of the ACM and IEEE, life fellow of IETE (India), and life member of ISTE (India).