Appl Intell (2015) 43:296–307
DOI 10.1007/s10489-014-0634-x

Engineering Simulated Evolution for Virtual Machine Assignment Problem

Sadiq M. Sait · Kh. Shahzada Shahid

Published online: 11 February 2015

Abstract Cloud computing is a rapidly growing services business in today's IT market. Its sharp growth is producing many challenges for cloud managers. One primary concern is to efficiently manage the cloud resources, i.e., to maximize utilization of hardware with minimum power consumption. Virtual Machine (VM) consolidation is a very helpful approach to achieve these goals. In this context, we investigate the VM assignment problem. We describe the engineering of a nondeterministic iterative heuristic known as Simulated Evolution (SimE) to solve the well-known NP-hard problem of assigning VMs to hardware hosts. A 'goodness' function which is related to the target objective of the problem is defined. It guides the moves and helps traverse the search space in an intelligent manner. In the process of evolution, VMs with high goodness value have a smaller probability of getting perturbed, while those with lower goodness value may be reallocated via a compound move. Results are compared with those published in previous studies, and it is found that the proposed approach is efficient both in terms of solution quality and computational time demand.

S. M. Sait (✉)
Department of Computer Engineering and Center for Communications and IT Research, Research Institute King Fahd University of Petroleum & Minerals, Dhahran 31261, Saudi Arabia
e-mail: sadiq@kfupm.edu.sa

Kh. S. Shahid
Department of Computer Engineering, King Fahd University of Petroleum & Minerals, Dhahran 31261, Saudi Arabia
e-mail: g201304230@kfupm.edu.sa

Keywords Combinatorial optimization · Evolutionary metaheuristic · Simulated evolution · Virtual machine placement · NP-hard · Nondeterministic algorithms.

1 Introduction

In recent years, cloud-based data centers have emerged as a popular choice for hosting and delivering IT services. Due to economies of scale and ease of accessibility, the IT industry is rapidly adopting the cloud computing paradigm [1]. Public-cloud market growth rate was recently forecasted to be 18.5 % in the Gartner report 2013 [2]. With its fast growing market size, its energy consumption is also increasing alarmingly. In 2010, electricity consumption by data centers was estimated to be 1.1–1.5 % of total electricity usage worldwide with an expectation of further growth [3, 4]. In data centers, energy is not only consumed for running the physical machines but also for cooling the infrastructure. It is estimated that energy consumption accounts for approximately 12 % of monthly operational expenditures of a typical data center [5]. Also, large-scale data centers are facing regulatory restrictions on energy usage by governmental agencies who are promoting green computing [6]. Hence, reducing energy consumption is a primary goal of today's data center operations.

Many techniques have been proposed to reduce the energy consumption of data centers. These techniques suggest better control of power distribution systems [7], efficient cooling systems [8], optimized computer hardware [9], virtualization technology [10], and load balancing mechanisms [11, 12]. It is known that turning off some unused machines by intelligently allocating the workload to the smallest number of hosts is an effective approach to reduce energy cost of a data center. For example, turning off a

single x86 server from a data center can save approximately $400 per annum [13]. This is because an idle machine (running with no load) consumes 60–70 % of its peak-load power consumption [11, 12, 14]. Researchers are trying to leverage this fact to save energy by optimizing virtual machine (VM) assignment, an NP-hard problem [15–18].

VMs that run on the same physical machine (PM) of course, share physical resources. Utilizing physical machines beyond a certain limit can cause significant performance degradation. Therefore, it is necessary to guarantee that, while minimizing the total number of PMs used, no PM gets utilized beyond a certain percentage of its maximum capacity. This is ensured by an appropriate upper limit on maximum utilization of PM resources. Due to multidimensional nature of VM requests, the assignment problem is very challenging. Each VM has its own CPU, memory, and bandwidth requirements. Similarly, every PM has a fixed capacity across each of these dimensions. Hence VM to PM assignment should minimize the total number of PMs used without violating these capacity constraints. VM assignment is formulated often as a *vector bin packing problem* (VBP) [19], where the VMs that are treated as objects (o) are packed into PMs that are treated as bins (b). The computational complexity of VBP is $O(b^o)$ [19]. Clearly, it is impractical to enumerate all possible solutions for a large number of VMs (objects). Even the one-dimensional version of this problem is NP-hard.

We present a Simulated Evolution (SimE) based heuristic to efficiently find a near-optimal solution in a reasonably short amount of time. Its performance is compared with that of another well known iterative heuristic, Simulated Annealing (SA), and with two popular constructive algorithms, the improved versions of First Fit Decreasing algorithm referred to here as $\mathbf{FFD_{imp}}$ and Least Loaded algorithm referred to here as $\mathbf{LL_{imp}}$, both proposed by Ajiro et al. [19]. Simulation results are presented that demonstrate the effectiveness of proposed algorithm compared to these approaches.

The remainder of this paper is organized as follows: In Section 2 we briefly discuss the background of the problem and overview of related work. Section 3 formally defines the problem. Section 4 explains our proposed approach. In Section 5 we provide comparison of the results with other heuristics and provide performance analysis. Finally, we conclude our discussion in Section 6.

2 Background and related work

Cloud computing provides three major types of services: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). In each of these, a user can request as much or as little computing resources as he desires. Efficient allocation of these requests to hardware resources can significantly improve operational cost through energy savings. Virtualization technology allows co-existence of multiple operating systems on a single PM. User requests are mapped to VMs with desired characteristics, and each VM, with its own operating system, works in an isolated environment while sharing the underlying machine's computing resources with other VMs. Figure 1 illustrates the benefits of virtualization and how optimal VM assignment helps to reduce the number of active PMs in a data center. Consider the situation illustrated in Fig. 1. Five applications are running on five different PMs (Fig. 1a). None of these applications is using more than 50% of the physical resources, resulting in huge resource wastage and unnecessary power consumption. With virtualization technology we can execute these applications in VMs, and these VMs can then be placed in a fewer number of physical machines that better utilize the available resources with reduced power cost (Fig. 1b). However, by making an optimal placement of these VMs, we can achieve even higher utilization with lower power consumption (Fig. 1c).

In the last few years, much effort has been made to find methods for optimal VM placement with the objective of minimizing the number of active PMs. This is to reduce both capital and operational cost, including power and maintenance. Jing Xu et al. proposed a modified genetic algorithm to simultaneously minimize the total power consumption, resource wastage and thermal dissipation using a fuzzy multi-objective cost function [17]. Breitgand et al. modeled the problem as a stochastic bin-packing problem using statistical multiplexing of physical resources [20]. Gao et al. presented VMPACS, a modified ant colony optimization algorithm which gives a Pareto set (non-dominated solutions) that minimizes total resource wastage and power consumption [16]. Recently, for large-scale machine reassignment and packing problems that have multiple resources, Masson et al. proposed a Multi-Start Iterated Local Search for Packing Problems (MS-ILS-PP) [21]. Karmer et al. used the concept of dynamic voltage/frequency scaling (DVFS) along with VM consolidation to further improve the energy efficiency. Their approach is based on trade-off between power and performance [22].

In the category of deterministic heuristics, Doddvula et al. proposed a Magnitude Classified algorithm for server consolidation. Their algorithm first classifies the workload based on their resource requirements and then places the complementary workloads [23]. Other works such as that by Jhawar et al. [24] and Shi et al. [25] also consider security constraints along with maximizing utilization.

First Fit Decreasing (FFD) and its variants are other common deterministic methods applied to find an approximate solution to the vector bin-packing problem [15]. FFD algorithm first sorts the VMs in decreasing order of their sizes and then places them in PMs according to First Fit (FF)

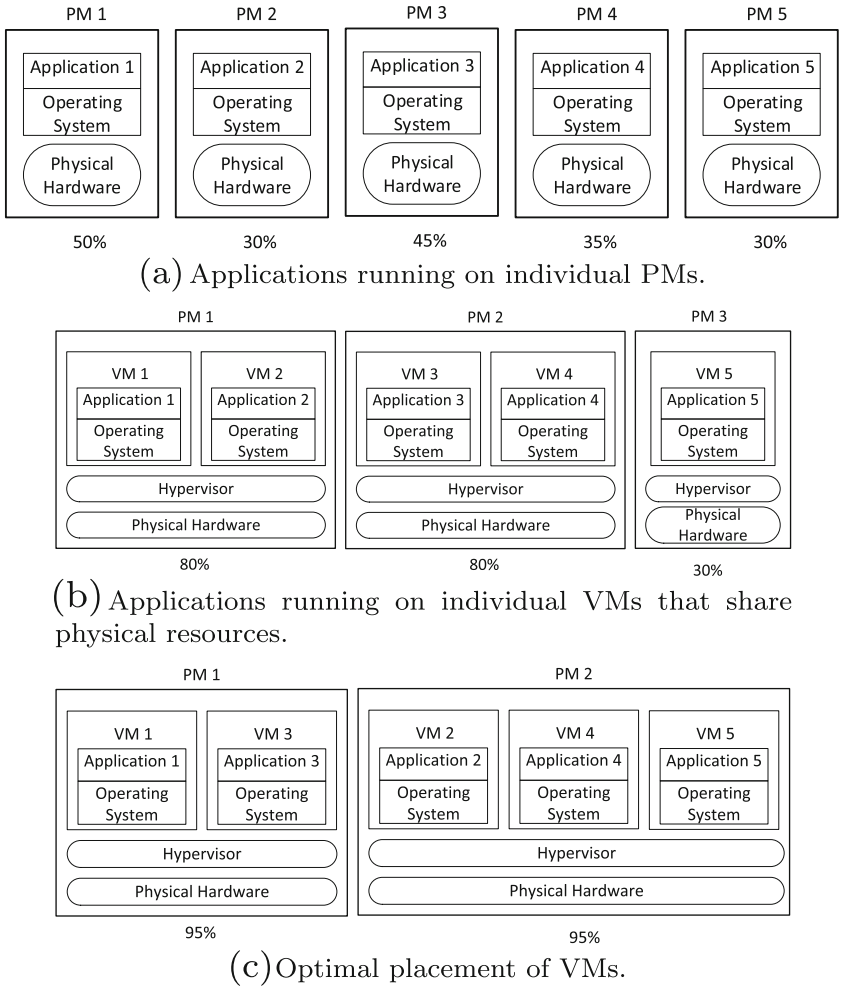


Fig. 1 The above diagrams show (**a**) poor utilization with 5 active PMs, (**b**) moderate utilization with 3 active PMs, and (**c**) optimal utilization with 2 active PMs only

strategy. In the context of data center energy optimization, Lei Shi et al. presented and evaluated the performance of six different FFD-based VBP algorithms [15]. Panigrahy et al. systematically studied the family of FFD heuristics and their limitations and suggested a new geometric based heuristic approach for VBP [18].

Ajiro et al. [19] suggested improvements to the classical FFD and least loaded (LL) algorithms. There are two differences between LL and FFD. First, LL restarts the placement process each time a new PM is added, when it has failed to place a VM into currently active PMs. Whereas FFD continues to place the subsequent VMs until all are placed. Second, in order to place a VM into a least-loaded active PM, LL sorts active PMs in ascending order of their current utilizations each time before placing it. Improved FFD (FFD_{imp}) is different from conventional FFD in the sense that it seeks near-optimal solution in multiple passes. In the first pass VMs are sorted in decreasing order of their highest resource demand, i.e., sum of all CPU demands or sum of all memory demands, whichever is larger. Then an attempt is made to pack all the VMs in number of PMs equal to the theoretical lower bound. If any VM cannot be packed, then it is moved to a priority queue and placement process is aborted. In the next pass, VMs in the priority queue are placed first, followed by the remaining VMs in the sorted list. The above steps are repeated MAXR times. If all VMs are not packed in MAXR iterations then the number of destination PMs is incremented by one and the above process is repeated until a solution is found, i.e., all VMs are packed. LL_{imp} is implemented in multi-passes in a similar way employing the LL heuristic. These improved versions provide better quality solutions than that of their single-pass implementations. This improvement, however, comes at the expense of increased run time [19].

In this work we engineer an evolutionary nondeterministic optimization heuristic known as Simulated Evolution (SimE). Similar to other nondeterministic algorithms, SimE also possesses hill-climbing capability. One key requirement of SimE is to define an appropriate way to estimate the *goodness* of the current assignment of a movable element, in our case the movable elements are VMs. The process of evolution, guided by goodness value, tends to converge

reasonably fast to a good quality solution. Many other non-deterministic heuristics, such as Simulated Annealing (SA), tabu search, etc., lack this domain knowledge feature and work mostly with random moves. Further details of goodness function developed for VM assignment are provided in Section 4.

3 Problem statement and formulation

In a VM consolidation problem, the goal is to place the VMs in the minimum number of possible PMs without performance degradation. And, turn off the remaining PMs to save power. When mutiple VMs are placed on a single PM, the host operating system or hypervisor may consume some extra resources, e.g., for resource scheduling, or context switching [24]. To avoid performance degradation of PMs, we set an upper-bound on the maximum utilization of any resource of a PM with some threshold value. This threshold value can be specified by the data center managers in terms of percentages.

In the context of vector bin-packing problems, VMs may be understood as objects that are packed into PMs that are considered bins. The size of each object, i.e., VM, is defined by its resource demand vector. Dimensions of these resources may include CPU, memory, bandwidth, disk space etc., as described in Section 1. Capacity of each bin, i.e., PM, is bounded by the selected utilization threshold of its resources. Also note that placing multiple VMs on the same PM has an additive effect on the PM's utilization across each dimension. For example, if (25 %, 35 %) is a pair of the CPU and memory utilization of a VM, and (20 %, 40 %) is of another VM, then, the utilization of a PM accommodating these two is (45 %, 75 %), i.e., the vector sum.

3.1 Optimization formulation

In this paper, we consider two dimensions of resources, CPU and memory. Suppose that there are n VMs to be assigned. Each VM v_i, $i \in \{1, 2, 3, \ldots, n\}$ is defined as a 2-dimensional requirement vector, $v_i = \{v_i^c, v_i^m\}$ where each dimension represents a normalized value of one type of resource requested (CPU and memory). These VMs are to be assigned to q PMs. Let T_k^c and T_k^m be the threshold values of CPU and memory resources, associated with each PM p_k, $k \in \{1, 2, 3, ..., q\}$ respectively. The assignment solution is represented by a $q \times n$ matrix A, where

$$A_{ki} = \begin{cases} 1 & \text{if VM}_i \text{ is assigned to PM}_k \\ 0 & \text{otherwise} \end{cases}$$

Let $f(p_k)$ be a function such that $f(p_k) = 1$, if PM p_k is loaded with at least one VM and $f(p_k) = 0$ otherwise. The problem of assigning all VMs to the least number of PMs, and subject to the constraints, can be formulated as follows:

$$\textit{minimize} \quad \sum_{k=1}^{q} f(p_k) \tag{1}$$

$$\textit{subject to} \quad \sum_{i=1}^{n} A_{ki}.v_i^c \leq T_k^c \quad \forall k \in \{1, 2, 3, ..., q\} \tag{2}$$

$$\sum_{i=1}^{n} A_{ki}.v_i^m \leq T_k^m \quad \forall k \in \{1, 2, 3, ..., q\} \tag{3}$$

$$\sum_{k=1}^{q} A_{ki} \leq 1 \quad \forall i \in \{1, 2, 3, ..., n\} \tag{4}$$

Constraints (2) and (3) impose threshold limit on maximum utilization while constraint (4) ensures that each VM will be assigned to only one PM.

4 Proposed approach

In this section we describe our Simulated Evolution (SimE) based VM consolidation algorithm. We begin with a brief discussion of the basic SimE heuristic.

4.1 Simulated evolution

Simulated Evolution (SimE) is a general iterative heuristic proposed by Kling and Banerjee [26]. This scheme combines iterative improvement and constructive perturbation and saves itself from getting trapped in local minima by following a stochastic approach. In SimE, the search space is traversed by making intelligent moves, unlike in other non-deterministic algorithms such as Simulated Annealing (SA), where random moves are made. The core of the algorithm is the goodness estimator. SimE assigns each moveable element a goodness value. The goodness value indicates how well a certain movable element is currently assigned. The higher the goodness value, the lower is the probability of the element being selected for reallocation.

The structure of the SimE algorithm is shown in the flowchart in Fig. 2. SimE assumes that there exists a solution Φ of a set V containing n movable elements (VMs). The algorithm starts from an initial assignment Φ_i, and then, following an evolution-based approach seeks to reach better assignments from one generation to the next by perturbing some ill-assigned elements (VMs) while retaining the near-optimal ones. The algorithm has one main loop consisting of three basic steps, **evaluation**, **selection** and **allocation**. The three steps are executed in sequence until the solution average goodness reaches a maximum value, or no noticeable improvement in solution quality is observed after a given number of iterations [27].

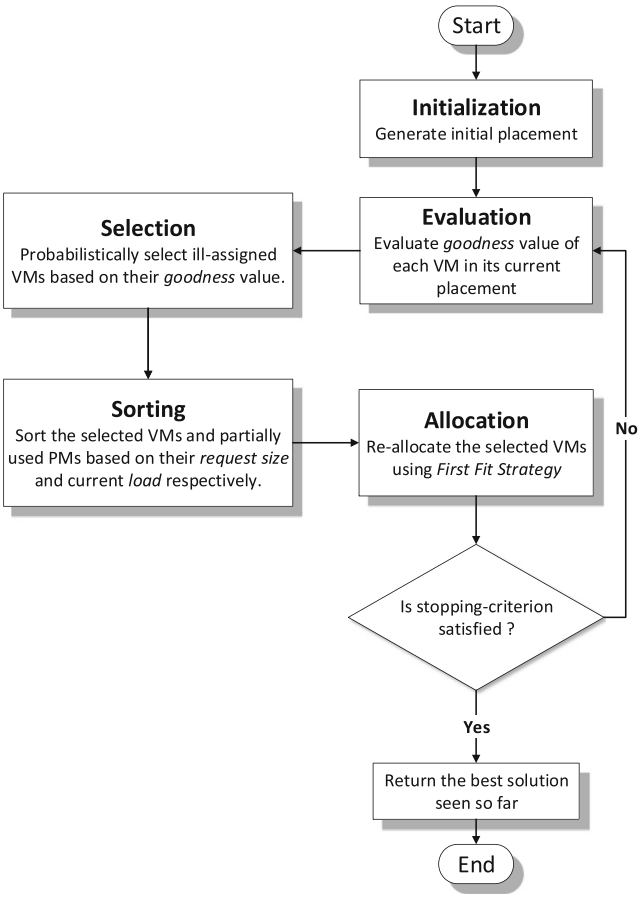


Fig. 2 Flowchart of SimE

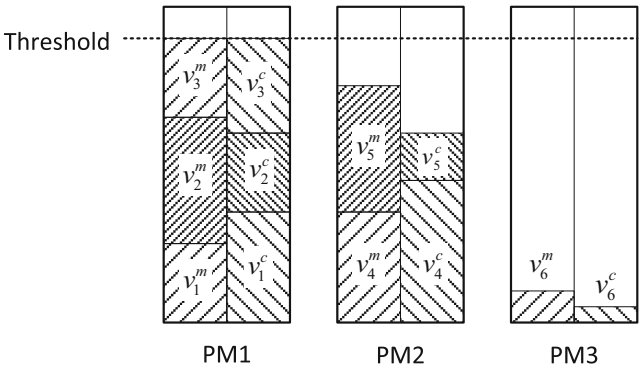


Fig. 3 Allocation of 6 VMs on 3 PMs

4.1.1 Goodness evaluation

The Evaluation step consists of evaluating the goodness (fitness) g_i of each VM v_i assigned to PM p_k in current solution Φ'. Effective goodness measures can be thought of based on the domain knowledge of the optimization problem [28]. The goodness measure must be a single number expressible in the range $[0, 1]$. For our VM assignment problem we define the goodness measure as:

$$g_i = \frac{v_i^c + v_i^m}{p_k^c + p_k^m}, \qquad p_k^c \leq T_k^c \ \& \ p_k^m \leq T_k^m \tag{5}$$

Where v_i^c and v_i^m are CPU and memory requirements of VM v_i, and p_k^c and p_k^m are the available CPU and memory resources of partially used PM p_k after removing VM v_i from PM p_k in the current solution Φ'. Equation (5) assumes a minimization of resource wastage in PM p_k (maximization of goodness). The goodness of a VM v_i will be 1 if it is assigned to such a partially used PM p_k that $v_i^c = p_k^c$ and $v_i^m = p_k^m$. It means that the current assignment of VM v_i exactly packs the PM p_k and hence optimally utilizes the PM p_k. For example, the goodness values of VMs v_1, v_2 *and* v_3 (in Fig. 3) are 1 as their combined placements optimally utilize the resources of PM p_1. On the other hand, the goodness g_i will be near 0, when a VM v_i, with a very small resource requirements, is placed in an empty PM p_k i.e., $v_i^c << p_k^c$ and $v_i^m << p_k^m$. Such an assignment will result in maximum resource wastage. The VM v_6, in (Fig. 3), has approximately zero goodness value. Note that this goodness estimation is strongly related to the target objective of the given problem. The quality of a solution can also be estimated by summing up the goodness of all of its constituent elements (VMs).

The goodness measure given in (5) can be generalized for D-dimensional case as in (6):

$$g_i = \frac{v_i^{d_1} + v_i^{d_2} + \ldots \ v_i^{d_D}}{p_k^{d_1} + p_k^{d_2} + \ldots \ p_k^{d_D}}, \quad p_k^{d_1} \leq T_k^{d_1}, \ p_k^{d_2} \leq T_k^{d_2}, \ldots, \ p_k^{d_D} \leq T_k^{d_D} \tag{6}$$

4.1.2 Selection

In this step, the algorithm probabilistically selects elements for reallocation. Elements with low *goodness* values have higher probabilities of getting selected. Selection step partitions Φ' into two disjoint sets; a selection set V_s and a partial solution Φ_p containing the remaining elements of the solution Φ'. Each element in the solution is considered separately from all other elements. The decision whether to assign an element v_i to the set V_s is based solely on its goodness g_i. The selection operator has a nondeterministic nature, i.e., an individual with a high goodness (close to one) still has a non-zero probability of being assigned to the selection set V_s. It is this element of nondeterminism that gives SimE the capability of escaping local minima. Each time a VM v_i is considered for selection a random number is generated. The inequality $Random \leq (1 - g_i + B)$ is used for this purpose (see Fig. 4). A selection *bias* (B) is used to compensate for errors made in the estimation of goodness. Its objective is to inflate or deflate the goodness of elements. A high positive value of bias decreases the probability

```
ALGORITHM
Simulated_Evolution(V, Stopping − criteria);
/* Φi: Initial Solution; */
/* Φp: Partial Solution; */
/* Φ′: New Solution; */
/* V: Set of all VMs, where |V| = n; */
/* Vs: Selected VMs for reallocation; */
/* Pa: Active PMs in Φp; */
/* B: Selection bias; */
/* maxSelection: Upper limit of the selection set size; */
INITIALIZATION;
    Φi = initial_placement(V);
    Φ′ = Φi;
Repeat
EVALUATION:
    ForEach vi ∈ V Do
        gi = Evaluate(vi); /* Evaluate using goodnes estimator (Equation 5) */
    EndForEach;
SELECTION:
    Φp = Φ′;
    counter = 0;
    ForEach vi ∈ V Do
        If (Random ≤ (1 − gi + B)) ∧ (counter ≤ maxSelection) Then
            Vs = Vs ∪ {vi};
            Φp = Φp − {vi};
            counter = counter + 1;
        EndIf;
    EndForEach;

ALLOCATION:
Sort the VMs in set Vs based on their resource demand (Equation (7));
Sort the active PMs Pa based on their current load (Equation (8));
    ForEach vi ∈ Vs Do
        Allocate( vi ,Φp); /* Allocate vi in Φp, using First Fit Strategy */
    EndForEach;
        Φ′ = Φp;
Until Stopping-criterion is satisfied;
Return (BestSolution);
End Simulated_Evolution.
```

Fig. 4 Simulated evolution algorithm for VM assignment

of selection while a negative value has the opposite effect. Large selection sets may lead to a better solution, but will require higher run time. On the other hand, small selection sets will speed-up the algorithm, increasing the risk of an early convergence to a sub-optimal solution (local minima). Values of B are recommended to be in the range $[-0.2, 0.2]$. In many cases a value of $B = 0$ would be a reasonable choice as in our case [27].

4.1.3 Allocation

Allocation is the SimE operator that has most impact on the quality of solution. Allocation takes as input the set V_s and the partial solution Φ_p and generates a complete new solution Φ' with the elements of set V_s mutated according to allocation strategy. The goal of *Allocation* is to favor improvements over the previous generation, without being too greedy [27]. Since the *goodness* of each individual element is also tightly coupled with the target objective, superior alterations are supposed to gradually improve the individual goodness as well. Hence, *Allocation* allows the search to progressively converge towards a configuration where each individual is optimally located.

The choice of a suitable allocation function is problem specific. Similar to the design of goodness function, the choice of allocation strategy also requires ingenuity on the part of the designer. In this work we adopted a variant of FFD heuristic as our allocation strategy. The VMs selected during the *selection* step are sorted in decreasing order of their request sizes (Rv_i) computed using equation (7).

$$Rv_i = (v_i^c)^2 + (v_i^m)^2 \tag{7}$$

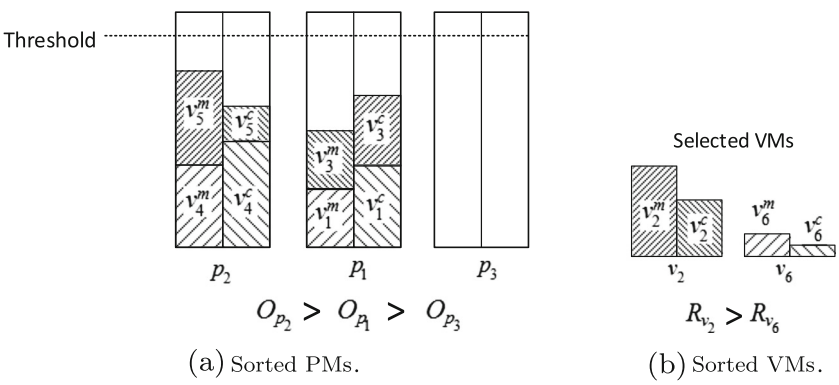
The active PMs in partial solution Φ_p are also sorted in decreasing order of the linear sum of their occupied resources (Op_k) computed using equation (8):

$$Op_k = (1 - {p_k}^c) + (1 - {p_k}^m) \tag{8}$$

Subsequently, First Fit algorithm is applied to generate the new solution Φ'.

To illustrate this, consider the placement solution in Fig. 3. Suppose that v_2 and v_6 are selected for reallocation. These VMs are sorted according to their size using equation (7) and PMs are sorted according to their utilized size using equation (8). This is illustrated in Fig. 5. In the next step, the first fit algorithm will first attempt to place VM v_2 in PM p_2. Since the remaining capacity of p_2 is not sufficient to accommodate v_2, it will be placed in next PM, that is p_1. Next v_6 will be attempted & successfully placed in p_2. This resulting solution is shown in Fig. 6. This new solution is better than the previous one as it requires one less PM, and it thereby results in energy saving.

Initial placement Φ_i is also obtained by this same *allocation* strategy but with the difference that all the VMs are treated as *selected* and are placed in an empty set of PMs.



(a) Sorted PMs. (b) Sorted VMs.

Fig. 5 Sorted PMs and VMs for allocation

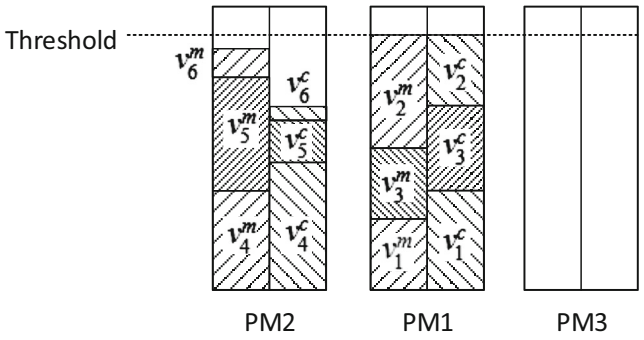


Fig. 6 Allocation of selected VMs

4.2 Complexity analysis

Our proposed SimE-based algorithm consists of four steps in a loop as illustrated in the flowchart in Fig. 2. The evaluation step computes *goodness* value of all n VMs using equation (5). This takes $O(n)$ time. The selection step probabilistically selects ill-assigned VMs and this also takes $O(n)$ time. In the sorting step prior to allocation, both the lists of selected VMs and active PMs are sorted and this takes $O(n\ log\ n)$ time. In the allocation step, First Fit (FF) algorithm sequentially checks if all selected VMs can be packed into one of q current active PMs. FF then packs each selected VM into a PM first found to be able to accommodate it. If a VM cannot be packed into any current active PM, the $(q+1)$-th PM is turned ON to accommodate it. The complexity of this step is $O(n^2)$. The overall complexity of our algorithm is $O(I.n^2)$, where I is the number of iterations. Experiments have indicated that I remains fairly constant as n increases, e.g., I varies in the range of 65–75 when n is increased from 200 to 1000.

5 Performance analysis

In this section we provide performance evaluation of our proposed approach with respect to solution quality and run time. First, we compare it with the improved versions of classic FFD (FFD_{imp}) and LL (LL_{imp}) algorithms proposed by Ajiro et al. [19], and a well-known iterative heuristic, Simulated Annealing (SA) [27]. Then we discuss the solution quality, performance, and scaling of the SimE heuristic.

5.1 Simulation setup

The programs for the proposed SimE algorithm, SA, FFD_{imp} and LL_{imp} heuristics were coded in MATLAB and run on an Intel $core^{TM}$i5 with 1.80 GHz CPU and 4 GB RAM. FFD_{imp} and LL_{imp} try to pack all the VMs in number of PMs equal to the theoretical lower bound (LB). If that can not be achieved then a different packing sequence based on reordered VMs (as explained in Section 2) is attempted before a new PM is turned ON. This is done upto a maximum of *MAXR* times, where MAXR is a control parameter that provides trade-off between quality of solution and time. Details of implementation and experiments to determine the appropriate value of MAXR are as discussed by Ajiro et al. [19]. According to the experimental study, MAXR for FFD_{imp} is set to 10–30 % and for LL_{imp} equal to 10 % of total number of VMs [19].

The details of Simulated Annealing (SA) algorithm can be found in [27]. SA has four important parameters which need to be tuned very carefully. These are: initial temperature T_0, cooling rate α, constant β, and M which represents the time until the next parameter update. After trial runs, appropriate values of these parameters were found to be $T_0 = 36$, $\alpha = 0.9$, $\beta = 1.1$, $M = 3$ for placement of 200 VMs. For 500 VMs, the best values of the parameters used were $T_0 = 120$, $\alpha = 0.9$, $\beta = 1.09$, $M = 2.2$. Our SimE algorithm was set to stop exploring the search space if no improvement was observed in the last 75 iterations. For SimE, *Bias* value was set to 0, and maximum size of selection set was restricted to $maxSelection = 40\,\%$ of total VMs (reasons are discussed in Section 4.1).

5.1.1 Work load

The problem instances were a set of two resource demand vectors representing the CPU and memory utilization of 200 and 500 VMs. PMs were assumed to be identical, that is, all PMs have the same resource capacity fixed at 90 % although the proposed approach is equally applicable for the heterogeneous case. Due to nondeterministic behaviour, average of results obtained from 100 independent runs are reported. In combinatorial problems, hardness is defined according to a worst-case scenario. However, in practical applications engineers invariably are more interested in typical instances of an optimization task rather than looking for the hardest possible instances. For this reason, suitably parametrized random ensembles of instances of problems are introduced. In this context, it was observed that in some regions of the ensemble space instances are typically easy to solve, while in other regions instances are found to be typically hard. This change in behaviour resembles the phase transitions observed in physical systems [29]. Two properties that determine the phase transition in our case are: (a) the correlation between the resource demand vectors (that is between CPU and memory sizes); and (b) average size of the VM resource demand in the data set. If the the CPU and memory utilization demands are almost equal for all VMs, then this is a special case of strong-positive correlation. Inthis case the instance is similar to a one-dimensional bin-packing problem that is rela-

For $i = 1$ **To** n **Do**
  $v_i^c = \text{rand}(2\overline{v^c})$
  $v_i^m = \text{rand}(\overline{v^m})$
  r = rand(1)
  If $(r < P \wedge v_i^c \geq \overline{v^c}) \vee (r \geq P \wedge v_i^c < \overline{v^c})$ **Then**
    $v_i^m = v_i^m + v^{\overline{m}}$
  EndIf;
EndFor;

Fig. 7 Pseudo code to generate random problem instances with certain correlations

tively easier to solve than the two-dimensional problem typically is [19].

However, for negatively correlated instances, more effort is required to find good solutions [18]. High negative correlation between CPU and memory requirements or high average size of the VM resource demands makes it difficult to find a solution near the theoretical lower bound. Such solutions either take more time, or for the same time the quality of results obtained is lower than those where the average size of the VM resource demands is small. To make synthetic instances more representative and cover a wide range of possible workloads, we generated problem instances with two different average resource values and several correlations of CPU and memory utilizations, employing the method proposed by Ajiro et al. [19]. The pseudo code for this is given in Fig. 7.

In Fig. 7, rand (1) is a function that returns uniformly distributed random real numbers in the range [0,1); $\overline{v^c}$ denotes the average CPU utilization while $\overline{v^m}$ represents the average memory utilization. The probability *P* is used to decide whether both the utilization of CPU and memory would be equal to or higher than the average values, or both utilizations would be lesser than the average values. By varying this probability *P*, we can control the correlations of CPU and memory utilizations to some extent.

In our experiment, we used two kinds of average values and five different probabilities. We set both $\overline{v^c}$ and $\overline{v^m}$ to 25 %, and then to 45 %. The distributions of CPU and memory utilizations were in the range of [0, 50 %) when

Table 1 Performance comparison of FFD_{imp}, LL_{imp}, SA and SimE

				VMs = 200			VMs = 500		
	Corr	Algorithm	MAXR	q	q/LB	Time(Sec)	q	q/LB	Time(Sec)
$\overline{v^c} = \overline{v^m} = 25\,\%$	Strong −ve	FFD_{imp}	20 %	68.67	1.1884	10.5987	171.95	1.2091	568.4928
		FFD_{imp}	30 %	65.68	1.1366	11.5814	163.90	1.1524	601.2377
		LL_{imp}	10 %	63.47	1.0982	2.4263	156.15	1.0978	68.5390
		SimE	–	59.22	1.0246	1.1137	145.00	1.0193	8.8654
		SA	–	70.29	1.2173	7.7961	182.45	1.2793	122.7781
	Weak −ve	FFD_{imp}	20 %	65.79	1.1411	8.0819	162.85	1.1489	385.2580
		FFD_{imp}	30 %	64.11	1.1118	9.5391	159.35	1.1242	471.4340
		LL_{imp}	10 %	62.24	1.0789	1.9123	152.25	1.0736	47.0452
		SimE	–	58.86	1.0203	0.9870	144.35	1.018	7.0985
		SA	–	69.89	1.2153	7.7669	180.55	1.2664	126.7085
	Zero	FFD_{imp}	20 %	63.25	1.1122	6.5057	160.40	1.1364	365.4011
		FFD_{imp}	30 %	62.51	1.0991	8.3291	158.95	1.1262	486.3127
		LL_{imp}	10 %	60.26	1.0594	1.4108	149.85	1.0615	35.8382
		SimE	–	57.89	1.0177	0.9166	143.40	1.016	5.9491
		SA	–	69.74	1.2179	7.7257	180.1	1.2717	131.3036
	Weak +ve	FFD_{imp}	20 %	61.80	1.0849	5.0155	156.75	1.1004	307.2129
		FFD_{imp}	30 %	61.62	1.0817	6.7264	156.35	1.0976	416.3680
		LL_{imp}	10 %	60.04	1.0535	1.1589	150.00	1.0529	30.6913
		SimE	–	57.84	1.0152	0.9330	144.30	1.013	6.8773
		SA	–	69.10	1.2098	7.6981	178.15	1.2596	126.4890
	Strong +ve	FFD_{imp}	20 %	60.44	1.0650	3.6218	149.70	1.0708	175.7809
		FFD_{imp}	30 %	60.40	1.0643	4.8976	149.70	1.0708	241.3745
		LL_{imp}	10 %	59.53	1.0488	1.0285	146.50	1.0479	23.8151
		SimE	–	57.63	1.0155	0.8121	141.50	1.0122	5.3684
		SA	–	67.86	1.1917	7.6885	172.85	1.2432	118.1567

Table 1 (continued)

				VMs = 200			VMs = 500		
	Corr	Algorithm	MAXR	q	q/LB	Time(Sec)	q	q/LB	Time(Sec)
$\overline{v^c} = \overline{v^m} = 45\,\%$	Strong −ve	FFD_{imp}	20 %	124.62	1.2065	28.35	300.20	1.1777	1330.0000
		FFD_{imp}	30 %	123.91	1.1996	39.4475	298.25	1.1701	1820.0000
		LL_{imp}	10 %	125.63	1.2163	8.6206	294.50	1.1553	464.8868
		SimE	–	121.47	1.1759	1.418	286.25	1.1229	10.5846
		SA	–	130.98	1.2634	10.1118	329.05	1.2885	154.7883
	Weak −ve	FFD_{imp}	20 %	123.16	1.1865	26.4677	301.05	1.1774	1260.0000
		FFD_{imp}	30 %	122.03	1.1756	35.6037	298.35	1.1669	1690.0000
		LL_{imp}	10 %	122.39	1.179	7.1789	293.50	1.1479	428.7857
		SimE	–	118.86	1.1449	1.4415	286.50	1.1205	10.0072
		SA	–	128.23	1.2436	9.5241	326.50	1.2766	150.9985
	Zero	FFD_{imp}	20 %	119.14	1.1618	21.7631	293.90	1.1534	1060.0000
		FFD_{imp}	30 %	118.31	1.1536	29.4344	291.40	1.1436	1420.0000
		LL_{imp}	10 %	118.82	1.1585	5.8913	287.45	1.1279	342.8841
		SimE	–	114.94	1.1205	1.3314	278.60	1.093	9.2724
		SA	–	126.36	1.2316	9.2873	318.25	1.2639	144.1075
	Weak +ve	FFD_{imp}	20 %	116.65	1.1361	17.3948	286.40	1.1263	798.6678
		FFD_{imp}	30 %	116.18	1.1316	24.1385	285.15	1.1214	1110.0000
		LL_{imp}	10 %	116.35	1.1331	4.8078	281.30	1.106	269.7920
		SimE	–	113.10	1.1013	1.2424	273.35	1.0746	8.5355
		SA	–	123.68	1.2103	9.1476	313.65	1.2395	141.7936
	Strong +ve	FFD_{imp}	20 %	112.74	1.1136	13.4092	278.65	1.1064	646.3755
		FFD_{imp}	30 %	112.15	1.1077	18.5815	277.10	1.1002	885.2129
		LL_{imp}	10 %	110.69	1.0932	3.1699	270.45	1.0738	151.7008
		SimE	–	108.10	1.0675	1.0909	264.60	1.0505	7.7824
		SA	–	119.59	1.1817	8.8983	303.25	1.2130	134.8037

$\overline{v^c} = \overline{v^m} = 25\,\%$, and [0, 90 %) when $\overline{v^c} = \overline{v^m} = 45\,\%$. For $\overline{v^c}$ and $\overline{v^m} = 25\,\%$, we set P equal to 0.00, 0.25, 0.50, 0.75, and 1.0, and for this the average correlation coefficients obtained are −0.7485, −0.3813, 0.0081, 0.3736, and 0.7493 for each set of instances. These coefficients correspond to strong-negative, weak-negative, no, weak-positive, and strong-positive correlations. The same values of P were used for $\overline{v^c}$ and $\overline{v^m} = 45\,\%$ and then the correlation coefficients were −0.7508, −0.3703, 0.0019, 0.3857, and 0.7476. Threshold values of both utilizations were kept at $T_k^c = T_k^m = 90\,\%$, $k \in \{1, 2, 3, ..., q\}$ throughout these experiments.

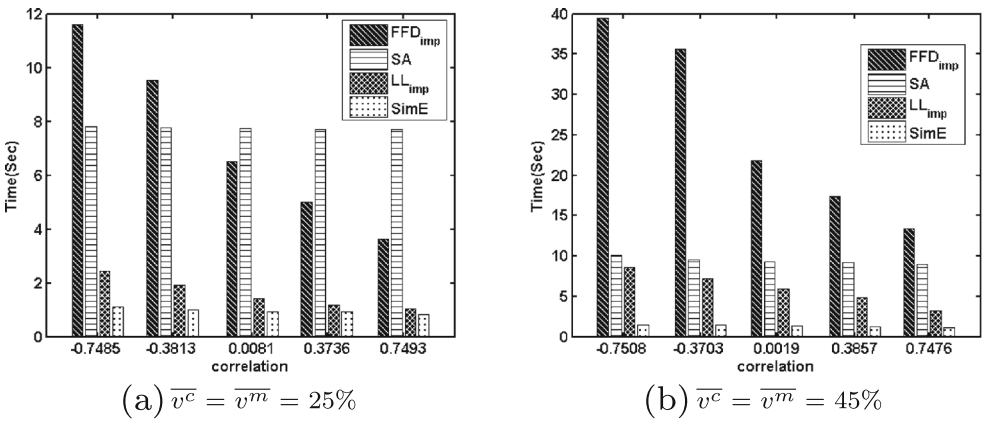


Fig. 8 Run time of FFD_{imp}, LL_{imp}, SA and SimE with 200 VMs for cases of (**a**) $\overline{v^c} = \overline{v^m} = 25\,\%$ and (**b**) $\overline{v^c} = \overline{v^m} = 45\,\%$

(a) $\overline{v^c} = \overline{v^m} = 25\%$ (b) $\overline{v^c} = \overline{v^m} = 45\%$

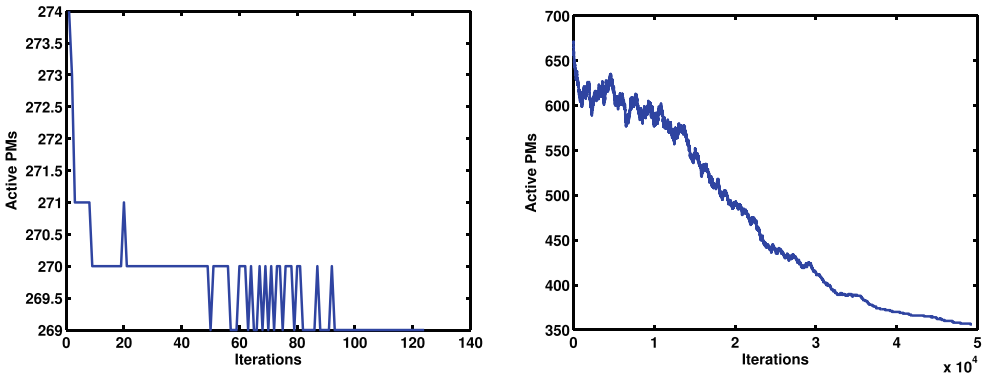


Fig. 9 Change in number of active PMs with iterations in (**a**) SimE and (**b**) SA

(a) SimE: Active PMs Vs Iterations. (b) SA: Active PMs Vs Iterations.

5.2 Results and discussion

To evaluate the efficiency of the proposed SimE algorithm, its performance is compared to that of FFD_{imp}, LL_{imp}, and SA. The comparison metrics are the number of active PMs (q), time to find the solution, and consolidation ratio (q/LB) calculated as ratio of active PMs q to the theoretical lower bound LB which is estimated using equation (9). A value q/LB closer to 1.0 represents higher efficiency. Table 1 lists the average number of active PMs obtained by these algorithms for different correlation and reference mean values.

$$LB = max\left(\lceil\sum_{i=1}^{n}\frac{v_i^c}{T^c}\rceil, \lceil\sum_{i=1}^{n}\frac{v_i^m}{T^m}\rceil\right) \tag{9}$$

From Table 1 we note the following:

- For all algorithms applied, consolidation ratio decreases with change of correlation from strong-positive to strong-negative.
- Similarly, consolidation ratio decreases by decreasing average resource demand value from 45 % to 25 %.
- The timing performance of both FFD_{imp} and LL_{imp} strongly depends on the correlation between CPU and memory utilizations. They take more time to reach a good solution for the instances with negative correlation (see Fig. 8). On the other hand, execution time of both SA and SimE only slightly varies across different correlations.
- In each case our proposed algorithm SimE gives better consolidation efficiency in a shorter amount of time as compared to FFD_{imp}, LL_{imp}, and SA.

SimE performs better than the two deterministic algorithms FFD_{imp} and LL_{imp} because these heuristics consider only one dimension, CPU or memory, the sum of whichever resource request is larger, when sorting them. The order obtained may not be suitable for optimal assignment. Therefore, the entire sorting and assignment steps are repeated several times, each time giving priority to VMs that have failed to be packed in currently active PMs. While our proposed SimE only picks a small number of VMs with low goodness value and sorts them considering both dimensions of utilization. This precise selection of a small number of VMs and proper sorting plays a key role in improving the solution quality and reducing run time. Although SimE and SA both are iterative nondeterministic heuristics, SimE is more intelligent, and thus requires fewer iterations to converge towards a desirable solution [27]. Change in cost of SimE and SA with iterations is illustrated in Fig. 9. It is clearly seen that SimE quickly finds a good solution through a few initial iterations. The plot of average goodness of the solution with iteration is shown in Fig. 10.

5.2.1 Scalability of SimE

In this subsection, we provide results of another set of experiments that are conducted to study whether the proposed algorithm is scalable to larger data centers and more VM

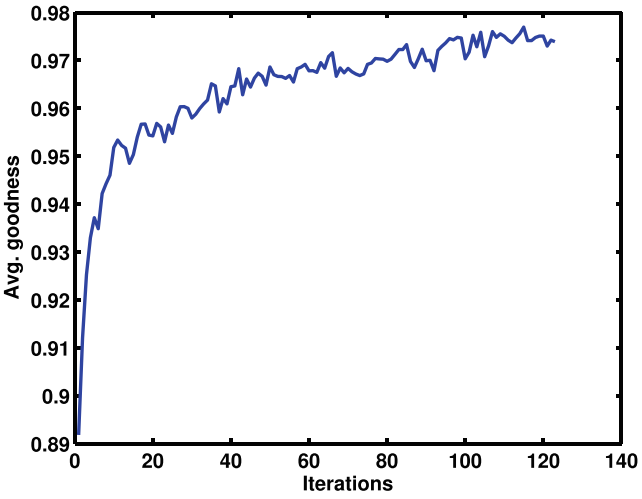
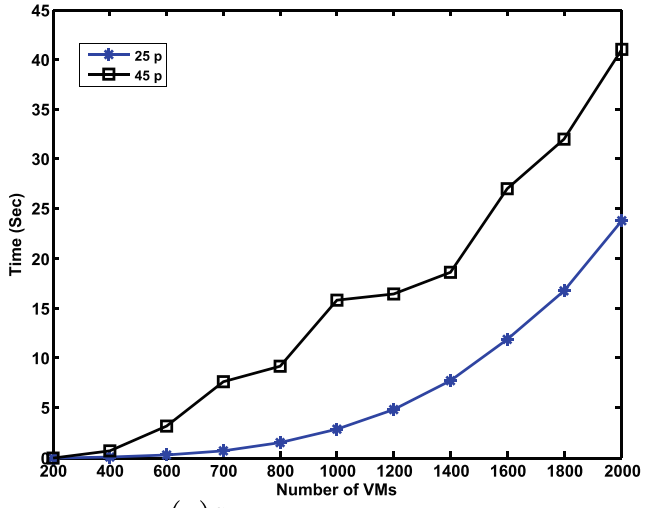
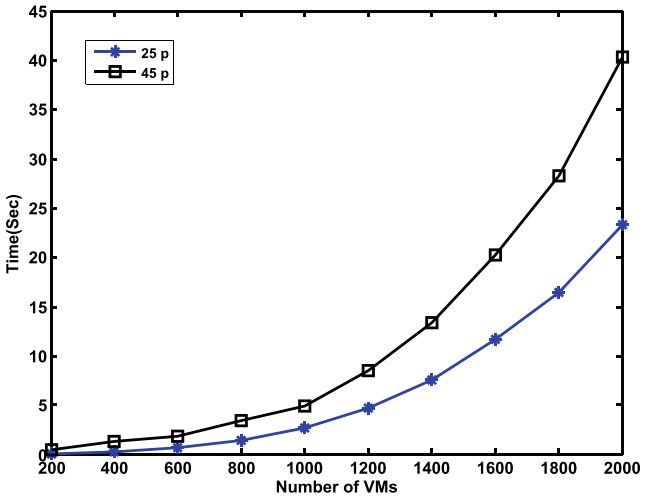


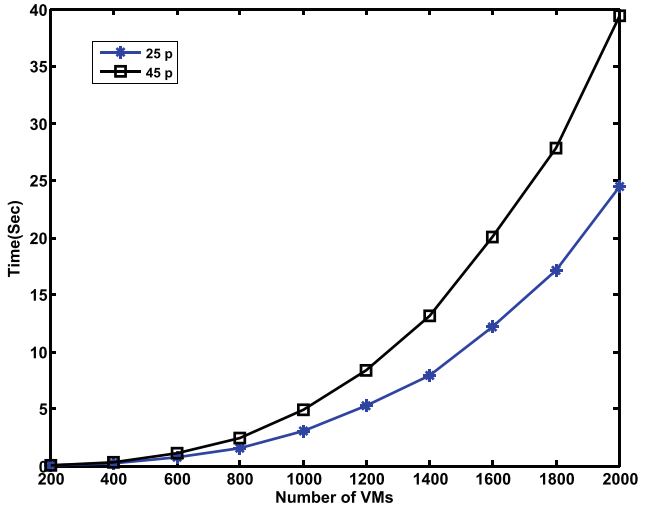
Fig. 10 SimE: change in the average goodness of VMs with iterations



(a) Strong negative correlation



(b) Zero correlation



(c) Strong positive correlation

Fig. 11 SimE algorithm run-time versus number of VM requests for different correlations

requests. In the experiment, the number of VMs is varied from 200 to 2000 for three different levels of correlations (strong-negative, zero and strong-positive correlations), and for two average resource demand values (25 % and 45 %). SimE was set to stop exploring the the search space when the consolidation ratio is reduced to lower than 1.07, or no improvement was observed in the last 35 iterations. The behaviour of the heuristic is shown in Fig. 11. The execution time is measured on Intel $core^{TM}2$ Quad with 2.67 GHz CPU and 4 GB RAM by taking an average of 50 runs. The algorithm takes less than a minute to solve a difficult assignment problem with up to 2000 VMs.

6 Conclusions and future work

A major concern for today's cloud service mangers is reducing energy consumption. This study investigated a multi-dimensional VM consolidation model to solve the problem. In this paper we presented the engineering of Simulated Evolution (SimE) search heuristic to find better solutions for the combinatorial NP-hard optimization problem, virtual machine assignment. Solutions in Simulated Evolution heuristic evolve based on the current goodness value of the assignments of VMs to PMs. We developed a goodness measure that enables SimE heuristic to quickly find the near-optimal solution. We evaluated its performance for a wide range of different problem instances. The important finding from this study is that the required run-time does not get affected by the correlation between different dimensions of VMs. This feature makes this heuristic desirable for all scenarios. In terms of consolidation efficiency, simulation results obtained are better than those published in literature and with savings in required computation time.

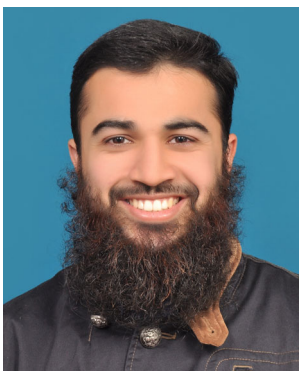
In this work we considered the scenario where all the VM requests are known before placement and the controller allocates them at once, trying to find the optimal allocation in accordance with the objectives and constraints. Such situations arise when a data center starts its operation after a maintenance state or when the data center optimizer/controller takes a decision at the back-end. However, in operational data centers VM requests arrive incrementally over time. In order to address this issue, it is recommended that future studies look into modifications of the algorithm that would work for an online scenario. In such cases existing VMs may have to be migrated for better allocation of new VMs.

Acknowledgments The authors acknowledge King Fahd University of Petroleum & Minerals (KFUPM) for all support. The work was conducted as part of project COE-572132-1. Special thanks to Ms. Fathima Chinoy and Dr. Blair Paul Bremberg for their help in editing and improving the quality of the manuscript.

References

1. Zhang Q, Zhani MF, Zhang S, Zhu Q, Boutaba R, Hellerstein JL (2012) In: Proceedings of the 9th International Conference on Autonomic Computing (ACM, 2012), pp 145–154
2. Gartner report (2013) http://www.gartner.com/newsroom/id/2352816
3. Koomey J (2011) Growth in data center electricity use 2005 to 2010. A report by Analytical Press, completed at the request of The New York Times
4. Beloglazov A (2013) Energy-efficient management of virtual machines in data centers for cloud computing. Ph.D. thesis, Department of Computing and Information Systems, The University Of Melbourne
5. Technology research - Gartner Inc. (2010) http://www.gartner.com/newsroom/id/1442113
6. Energy star computers specification (2012) http://www.energystar.gov/sites/default/files/specs//private/ES_Computers_Draft_1_Version_6.0_Specification.pdf
7. Raghavendra R, Ranganathan P, Talwar V, Wang Z, Zhu (2008) In: ACM SIGARCH Computer Architecture News (ACM, 2008), vol 36, pp 48–59
8. Bash CE, Patel CD, Sharma RK (2006) In: Thermal and Thermomechanical Phenomena in Electronics Systems, 2006. ITHERM'06. The Tenth Intersociety Conference on (IEEE, 2006), pp 445–452
9. Von Laszewski G, Wang L, Younge AJ, He X (2009) In: Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on (IEEE, 2009), pp 1–10
10. Verma A, Dasgupta G, Nayak TK, De P, Kothari R (2009) In: Proceedings of the 2009 Conference on USENIX Annual Technical Conference (USENIX Association, 2009), p 28
11. Chen G, He W, Liu J, Nath S, Rigas L, Xiao L, Zhao F (2008) In: NSDI, vol 8, pp 337–350
12. Fu Y, Lu C, Wang H (2010). In: Parallel & Distributed Processing (IPDPS),IEEE International Symposium on (IEEE,2010), pp 1–11
13. Technology research - Gartner Inc. (2009) http://www.gartner.com/newsroom/id/1234513
14. Gulati A, Holler A, Ji M, Shanmuganathan G, Waldspurger C, Zhu X (2012) VMware Tech J 1(1):45
15. Shi L, Furlong J, Wang R (2013) In: Computers and Communications (ISCC), 2013 IEEE Symposium on (IEEE, 2013), pp 000,009–000,015
16. Gao Y, Guan H, Qi Z, Hou Y, Liu L (2013) J Comput Syst Sci 79(8):1230
17. Xu J, Fortes JA (2010) In: Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on & Int'l Conference on Cyber, Physical and Social Computing (CPSCom) (IEEE, 2010), pp 179–188
18. Panigrahy R, Talwar K, Uyeda L, Wieder U (2011) Heuristics for vector bin packing. research. microsoft. com
19. Ajiro Y, Tanaka A (2007) In: Int. CMG Conference, pp 399–406
20. Breitgand D, Epstein A (2012) INFOCOM. In: Proceedings IEEE (IEEE, 2012), pp 2861–2865
21. Masson R, Vidal T, Michallet J, Penna PHV, Petrucci V, Subramanian A, Dubedout H (2013) Expert Syst Appl 40(13):5266
22. Kramer HH, Petrucci V, Subramanian A, Uchoa E, Comput Ind Eng (2012) 63(3):652
23. Doddavula SK, Kaushik M, Jain A (2011) In: Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on (IEEE, 2011), pp 332–339
24. Jhawar R, Piuri V, Samarati P (2012) In: CSE, pp 170–177
25. Shi L, Butler B, Botvich D, Jennings B (2013) In: Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on (IEEE, 2013), pp 499–505
26. Kling RM, Banerjee P (1987). In: Proceedings of the 24th ACM/IEEE Design Automation Conference (ACM, 1987), pp 60–66
27. Sait SM, Youssef H (1999) Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems (IEEE Computer Society Press)
28. Sait SM, Youssef H (1994) VISI physical design automation: Theory and practice. McGraw-Hill, New York
29. Hartmann A (2005) Phase transitions in combinatorial optimization problems - Basics, Algorithms and Statistical Mechanics. Wiley-VCH, New York

Sadiq M. Sait obtained a Bachelor's degree in Electronics from Bangalore University in 1981, and Master's and PhD degrees in Electrical Engineering from King Fahd University of Petroleum & Minerals (KFUPM), Dhahran, Saudi Arabia in 1983 & 1987 respectively. Since 1987 he has been working at the Department of Computer Engineering where he is now a Professor.In 1981 Sait received the best Electronic Engineer award from the Indian Institute of Electrical Engineers, Bangalore (where he was born). In 1990, 1994 & 1999 he was awarded the 'Distinguished Researcher Award' by KFUPM. In 1988, 1989, 1990, 1995 & 2000 he was nominated by the Computer Engineering Department for the 'Best Teacher Award' which he received in 1995, and 2000. Sait has authored over 200research papers, contributed chapters to technical books, and lectured in over 25 countries. Sadiq M. Sait is the principle author of the books (1) VLSI PHYSICAL DESIGN AUTOMATION: Theory & Practice, published by McGraw-Hill Book Co., Europe, (and also co-published by IEEE Press), January 1995, and (2) ITERATIVE COMPUTER ALGORITHMS with APPLICATIONS in ENGINEERING (Solving Combinatorial Optimization Problems): published by IEEE Computer Society Press, California, USA, 1999. He was the Head of Computer Engineering Department, KFUPM from January 2001 - December 2004, Director of Information Technology and CIO of KFUPM between 2005 and 2011, and now is the Director of the Center for Communications and IT Research at the Research Institute of KFUPM.



Kh. Shahzada Shahid is an MSc. student in Computer Engineering department at King Fahd University of Petroleum & Minerals. He obtained his Bachelor's degree in Electrical Engineering from UET Lahore in 2012. He worked as a research assistant in Al-Khawarzmi Institute of Computer Science for one year in Pakistan. His area of interests includes cloud computing, combinatorial optimization and heuristic algorithms.