Energy and cost-aware virtual machine consolidation in cloud computing

Amin Yousefipour1 | Amir Masoud Rahmani1,2 | Mohsen Jahanshahi3

1Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran
2Computer Science, University of Human Development, Sulaymaniyah, Iraq
3Department of Computer Engineering, Central Tehran Branch, Islamic Azad University, Tehran, Iran

Correspondence
Amir Masoud Rahmani, Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran; or Computer Science, University of Human Development, Sulaymaniyah, Iraq.
Email: rahmani@srbiau.ac.ir

Summary
Cloud computing has become an essential part of the computational world, offering a variety of server capabilities as scalable virtualized services. Big data centers that deliver cloud computing services contain thousands of computational nodes that consume a significant amount of energy. By introducing the virtual machine (VM), virtualization technology is trying to overcome this problem. One impressive technique for minimizing the total number of active physical servers that lead to improved energy consumption is VM consolidation. To optimize the consolidation process, effective VM placement can be used. In this paper, we first present a mathematical model aimed at reducing power consumption and costs by employing an effective VM consolidation in the cloud data center. Subsequently, we propose a genetic algorithm–based meta-heuristic algorithm, namely, energy and cost-aware VM consolidation for resolving the problem. Finally, we compare our proposed model with the well-known first fit, first fit decreasing, and permutation pack algorithms. The experimental results show that our proposed model reduced power consumption and costs when compared with the three demonstrated algorithms.

KEYWORDS
cost, cloud computing, power consumption, virtual machine, VM consolidation, VM placement

1 | INTRODUCTION

Cloud computing1 is currently recognized as a major solution in the IT industry by which scalable and elastically computation capabilities are delivered to users via services. Several large infrastructure companies2-5 provide their services through the cloud environment in which the computational resource (ie, processor, memory, and storage) can be accessed through the network. There are various paradigms of these services in the form of infrastructure as a service, platform as a service, and software as a service.6 In this paper, we focus on infrastructure-as-a-service clouds. The users can use these services in terms of service level agreement (SLA). The SLA is defined as a contract between cloud service providers and the users in which cloud service providers guarantee their provided services. It contains a number of service performance metrics such as the throughput, response time, storage space, and availability. If the providers are unable to satisfy the desired level of QoS requirements, they must pay penalties to the users.

Energy efficiency acts as a key issue in large-scale cloud environments.7 Data centers involve thousands of computational nodes that consume large amounts of electrical power. The power consumption of data centers has increased by 56% worldwide from 2005 to 20108; and the Gartner group predicts that, with these trends, energy costs will reach up to 50% of IT budgets in the next few years.9,10 Overlay power consumption has created many challenges such as the emission...
of large amounts of CO₂, rising cooling costs,¹¹ performance degradation,¹² and falling reliability.¹³ The power consumption of the data centers not only depends on the number of computational nodes and hardware power inefficiency but also on resource usage.

A major challenge for energy efficiency in cloud computing is the management of resources such as servers and virtual machines (VMs). Data centers pay 40%-50% of their power bills to run servers.¹⁴ As the workload increases, server power increases linearly. Furthermore, idle servers can consume about 60% of their peak power.¹⁴-¹⁶ Therefore, the number of idle servers should be reduced and more workloads should be added to running servers. The VMs are the primary computing blocks in cloud that allow physical machines (PMs) to efficiently run different workloads belonging to different users. With the provision of multiple VMs on a server, virtualization technology helps to manage resources as one of the core features of cloud. Server consolidation is a software technique for VM management that affects energy efficiency. The VM consolidation tries to execute multiple VMs on a small number of PMs so as to minimize the number of active PMs. Therefore, idle nodes can be switched off or function on standby mode to save energy.

In this paper, we model and formulate a VM placement problem as a mixed integer nonlinear programming (MINLP) problem. The proposed model considers the trade-off between power consumption and cost. The most important contributions of this paper are as follows:

• proposing an energy-efficient model for optimized placement of VMs in cloud environments;
• proposing a cost-efficient model for the optimized placement of VMs; and
• proposing a GA-based meta-heuristic algorithm for solving the optimized placement of VMs.

The remainder of the paper is organized as follows. In Section 2, related works are discussed. Section 3 presents the system model used in our work. Following the proposed model introduced in Section 3, the proposed meta-heuristic algorithm is investigated in Section 4. Experimental results and evaluation of the proposed algorithm are presented in Section 5. Finally, Section 6 presents the conclusions and future work.

2 | RELATED WORK

In cloud computing, power management has been studied extensively with the objective of minimizing the total power consumption in data centers. To this aim, researchers have proposed a wide range of approaches ranging from software-based techniques (such as server consolidation) to hardware-based solutions (such as DVFS). In this paper, we focus on software-based optimizations and, in particular, on effective VM placement techniques for server consolidation.

Sharifi et al. introduced an energy-aware scheduling algorithm to assign a set of VMs on a set of PMs with the aim of maximizing resource utilization and saving power. The proposed algorithm uses a set of objective functions based on fitness metric. To identify the performance interference between disk and processor utilization and the costs of migrations on VMs, they presented 4 models, ie, the target system model, the application model, the energy model, and the migration model. To consider more complex scenarios, where batch application running on VMs, Salimi and Sharifi presented an algorithm for scheduling a set of VMs on a shared PM. The goal of the algorithm is defined as minimizing the execution times of batch application running on VMs while considering interference of concurrent VMs. For identifying the interference, the interference model in terms of the number of concurrent VMs, processing utilizations of VMs, and network latency are presented. The authors assume that the VM and PM are homogenous and the capacity of each VM is fixed at runtime.

Tordsson et al. investigated the problem of the optimized placement of VMs across multiple cloud environments. The proposed approach uses a set of user-specified criteria including maximum budget and minimum performance and constraints such as load balance and hardware configuration of the individual VMs. They have explored a heterogeneous environment in which a fixed number of predefined hardware configurations of the VMs are considered. Their formulated problem is based on integer programming and enables price-performance placement tradeoffs. A brokering mechanism for deploying services across multiple cloud providers is proposed by Lucas-Simarro et al. which objectives at optimizing VM placement. In this approach, scheduling algorithm selection is based on optimizing some parameters such as total performance or total cost of the infrastructure in broker.

Beloglazov and Buyya investigated the problem of dynamic consolidation of VMs using live migration and switching idle nodes into sleep mode while meeting SLA requirements. To minimize the power consumption of the data center, an adaptive heuristics algorithm in terms of an analysis of historical data from resource usage by VMs are used.
Horri et al. proposed a QoS-aware VMs consolidation approach, which goals at reducing energy consumption and at the same time considering SLA violations in some cases. To optimize the VM placement, they consider the trade-off between energy consumption and performance. The authors demonstrate that the number of VMs on the host plays an important role in detecting under-loaded hosts. Moreover, they indicated that using the resource utilization history of VMs in the process of resource management in data centers can reduce energy consumption and SLA violations.

Dashti and Rahmani proposed a hieratical architecture to satisfy both provider and consumer requirements. In the proposed architecture, they attempted to improve the energy efficiency of VM allocation by using a modified particle swarm optimization algorithm. They balanced overloaded hosts by reallocating migrated VMs. In addition, to provide power saving, they also dynamically consolidated the under-loaded host. To provision VMs’ cost effectively, a novel learning-based resource provisioning has been proposed. The proposed approach focuses on how to lower the resource provisioning cost while not severely degrading the performance metrics of services.

3 PROBLEM MODELING

This section starts with the presentation of the target system model of cloud computing environment considered in this paper, followed by a detailed description of the power model of cloud data centers. We then focus on the cost model.

3.1 System model

The model of the target system is shown in Figure 1. As seen in this figure, there are 3 major roles in the system model including user, cloud provider, and cloud broker. The cloud broker is the main part of the system model, which acts as an intermediary between the user and providers to handle the use and delivery of cloud services, considering the performance requirements. The cloud broker can provide the system transparency in which the cloud providers are invisible to the users and a user interacts with the broker instead of directly communicating with a provider.

As represented in Figure 1, a user requests a virtual infrastructure by specifying the required information in the form of a request description template. This template can contain information such as optimization criteria (e.g., total price, execution time), a set of constraints, the number of VMs to be deployed on available clouds (e.g., computing or storage

FIGURE 1 The system model. VM, virtual machine [Colour figure can be viewed at wileyonlinelibrary.com]
resources), and some restrictions about the maximum number of a certain instance type. The request description template is delivered to the cloud scheduler, which is a component of the broker.

The scheduler is responsible for optimizing the VM placement based on user requirements and provider offers. Within the scheduler, an optimization model to minimize power consumption and cost can be applied. The output of the scheduler called placement plan, which is the most suitable placement of VMs regarding the specified requirements, is delivered to the virtual infrastructure manager component. The infrastructure manager can be implemented by a software platform such as OpenStack, OpenNebula, or Eucalyptus to control and manage cloud resources and handle user requests.

The virtual infrastructure manager component periodically asks the registry component about any updates on the availability of resources and their prices. The registry obtains the required information from each particular cloud provider involved in the cloud environment. A new provider, which is interested in joining the environment, registers its own information in the registry by sending a register message to the registry. The information of the newly added provider can be used in the next round of the decision-making process about VM placement. Since a lifetime has not been defined for the registration of a provider, it is not required for the provider to re-register until it decides to leave the environment or failure occurs. When a provider decides to leave the environment, it sends an unregister message to the registry, which causes the information of the provider to be removed from the registry. In this case, the scheduler does not use the removed provider in the decision-making process for the next VM placement phase.

We assume that each user request is encapsulated in a VM, which may have different sizes and configurations called instance types. Each cloud provider is comprised of a number of servers or PMs that are responsible for hosting these VMs. In this paper, each server supports at most 3 different instance types, namely, small, medium, and large. Figure 2 represents a cloud provider with 4 servers, each containing a variety of VMs.

3.2 Power model

In general, various factors affect the power consumption of each data center. These include cooling equipment, communication infrastructure, physical servers, and so on. In this work, the total power consumption of physical servers is considered as the power consumption of data centers, whereas other factors are ignored. Additionally, the power consumption of each server is calculated by considering 2 states of the server, ie, static state and running state. The static state represents an idle state in which no VM exists on the server and the server is active. The running state involves the allocation process of VMs. Therefore, $P_i$ is the power consumption of a server $s_i$ at a data center that can be modeled by a power function as

$$P_i = P_{i\text{idle}} + P_{i\text{placement}} + P_{i\text{vm}} + P_{i\text{switch}},$$

(1)
where $p_{i \text{idle}}$ is the power consumption of server $s_i$ in the idle state. Moreover, $p_{i \text{placement}}$ is the power consumed by running different VM instances on server $s_i$ because of the mapping of some requested VMs on this server based on the placement plan. The resulting outcome on empirical experiments show that running a new VM on a server without processing the workload leads to increasing the power consumption of the server in the idle state. This change in value is based on a nonlinear function. For simplicity, it is assumed that a constant value per new VM is added to the power consumption of the server in the idle state. The $P_{i \text{vm}}$ denotes this constant value in a server $s_i$. $p_{i \text{switch}}$ is the power consumption imposed to the system when a server switches between on and off states. In this paper, since the servers are considered to be homogeneous, the power consumed during switches are the same for all servers.

### 3.3 Cost model

In the proposed model, the cost is the user-paid fee for renting a VM with a specific type on a specified server within an hour. Hence, the cost of a server is estimated as the sum of the costs of all instance types running on it. Since servers, like VMs, can be presented in a variety of configurations, the cost of running a VM is different on each server. This difference can be due to different reasons such as the geographic location of servers and the hardware and technology used in it. In addition, servers may not support all instances of VMs. For example, as shown in Figure 2, Server 4 offers all instance types, whereas Server 2 only supports small and medium VMs.

### 3.4 Problem formulation

The VM placement problem can be addressed using either a static or dynamic approach. Since the number of the VMs required to respond a user request is considered to be constant in this research, the decision on VM placement is made once at a time point in each predefined time interval. In this case, the optimization algorithm runs periodically to adopt the VM deployment. Regarding static VM placement, our model presents a multiobjective model for power and cost optimization in a cloud environment based on mixed-integer nonlinear programming. In the proposed model, we first minimize the total power consumption of consolidated VMs subject to minimizing the number of servers required to host VMs. The total power consumption of the infrastructure is defined as the sum of the power consumption of all servers that host VMs in a cloud provider. After minimizing the total power consumption that can be considered as a system-oriented measure, we change the goal function and try to minimize the cost of the VMs placement, which is a user-oriented objective. Finally, we try to optimize more than one criterion at a time, making our problem a multiobjective optimization one. Therefore, 2 objectives are combined using the weighted sum method and a single-objective function, which simultaneously minimizes both total power consumption and cost, is presented.

More precisely, the objective of the proposed model is mapping a set of $N$ requested VMs, $V = \{v_1, v_2, \ldots, v_N\}$, to a set of $M$ available servers, $P = \{p_1, p_2, \ldots, p_M\}$, to minimize power consumption in the first stage. The servers are assumed to host a variety of possible VM instance types based on their resource capacities (e.g., the number of processor cores, memory, disk storage, etc). The instance types are denoted by $I_1, \ldots, I_L$, where $L$ is the number of all accessible instances. Let $P_{i k}$ represent the maximum power consumed to run a VM of instance type $l_k$ on server $s_i$, where $1 \leq i \leq M$. Without loss of generality and to simplify the model, the power consumption of the server running $w$ concurrent VMs is assumed to be the sum of power consumptions of $w$ VMs when running in isolation. This assumption helps to find the VM placement, which minimizes the power consumption. After finding the suitable placement, we use a formula to estimate the real power consumption according to the results gained from the observation, as will be mentioned in Section 5. Let $R_{i \text{CPU}}^j$, $R_{i \text{Mem}}^j$, and $R_{i \text{Disk}}^j$ denote the CPU, memory, and disk requests of a VM with instance type $j$, respectively. Similarly, server $s_i$ has CPU, memory, and disk capacities represented by $C_{i \text{CPU}}$, $C_{i \text{Mem}}$, and $C_{i \text{Disk}}$, respectively.

We define the Boolean variable $x_{ijk}$ that is 1 if the instance type $I_j$ of the server $s_i$ is selected to host VM $v_k$; otherwise, it is 0. Furthermore, the decision variable $y_i$ is set to 1 if the server $s_i$ hosts at least one VM, and 0 otherwise. After VM placement, if the server $i$ is turned off, the values of variables $y_i$ and $y'_i$ are 1 and 0, respectively. Otherwise, if the server $i$ is turned on, the values of these variables are 0 and 1. All the variables and input parameters used in the model are represented in Table 1 for ease of reference.
### TABLE 1 Notations used in the model

**Indices**

- \(i\): Server index, \(i=1,2,\ldots,M\)
- \(j\): VM instance type index, \(j=1,2,\ldots,L\)
- \(k\): VM index, \(k=1,2,\ldots,N\)

**Parameters**

- \(P_{ij}\): Power consumption of each VM instance type \(j\) on server \(i\)
- \(P_{i}^{\text{idle}}\): Idle power consumption on server \(i\)
- \(R_{j}^{\text{CPU}}\): CPU requests of each VM instance type \(j\)
- \(R_{j}^{\text{Mem}}\): Memory requests of each VM instance type \(j\)
- \(R_{j}^{\text{Disk}}\): Disk requests of each VM instance type \(j\)
- \(C_{i}^{\text{CPU}}\): CPU capacity of each server \(i\)
- \(C_{i}^{\text{Mem}}\): Memory capacity of each server \(i\)
- \(C_{i}^{\text{Disk}}\): Disk capacity of each server \(i\)
- \(L_{ij}\): Number of VM instance types \(j\) on server \(i\)
- \(C_{ij}\): Price of running VM instance type \(j\) on server \(i\)
- \(y_{i}'\): 1, if new state of server \(i\) is on, otherwise 0
- \(y_{i}\): 1, if current state of server \(i\) is on, otherwise 0
- \(Z\): A very large number
- \(P_{i}^{\text{Switch}}\): Power consumption of turning on or off the server \(i\)

**Variables**

- \(x_{ijk}\): 1, if VM \(k\) can be done on VM instance type \(j\) of server \(i\), otherwise 0
- \(y_{i}\): 1, if at least one VM instance type runs on server \(i\), otherwise 0

Abbreviations: VM, virtual machine.

The MINLP proposed model to optimize the power consumption can be formulated as

\[
F1 : \quad \sum_{i=1}^{M} \left( \sum_{j=1}^{L} \sum_{k=1}^{N} P_{ij} x_{ijk} + P_{i}^{\text{idle}} + \sum_{j=1}^{L} \sum_{k=1}^{N} x_{ijk} \right) y_{i} + \sum_{i=1}^{M} \left| y_{i} - y_{i}' \right| P_{i}^{\text{Switch}}. \tag{2}
\]

Subject to

- \(\sum_{i=1}^{M} \sum_{j=1}^{L} x_{ijk} = 1, \quad \forall k \in \{1, 2, \ldots, N\}\) \tag{3}
- \(\sum_{j=1}^{L} \sum_{k=1}^{N} R_{j}^{\text{CPU}} x_{ijk} \leq y_{i} \cdot C_{i}^{\text{CPU}}, \quad \forall i \in \{1, 2, \ldots, M\}\) \tag{4}
- \(\sum_{j=1}^{L} \sum_{k=1}^{N} R_{j}^{\text{Mem}} x_{ijk} \leq y_{i} \cdot C_{i}^{\text{Mem}}, \quad \forall i \in \{1, 2, \ldots, M\}\) \tag{5}
- \(\sum_{j=1}^{L} \sum_{k=1}^{N} R_{j}^{\text{Disk}} x_{ijk} \leq y_{i} \cdot C_{i}^{\text{Disk}}, \quad \forall i \in \{1, 2, \ldots, M\}\) \tag{6}
- \(\sum_{i=1}^{M} \sum_{j=1}^{L} \sum_{k=1}^{N} x_{ijk} = N, \quad \forall j \in \{1, 2, \ldots, L\}\) \tag{7}
- \(\sum_{i=1}^{M} \sum_{j=1}^{L} x_{ijk} \leq L_{ij}, \quad \forall j \in \{1, 2, \ldots, L\}\) \tag{8}
The objective function \( F_1 \) consists of several power consumption terms. It seeks to minimize the sum of the idle, placement, VM, and switching powers. The constraint mentioned in (3) ensures that each VM is hosted by exactly one instance type provided by the specific server. Constraints (4)-(6) checks that the CPU, memory, and disk usages of all VMs allocated on each server \( s_i \), \( 1 \leq i \leq M \), do not exceed the total capacities of server \( s_i \). Constraint (7) guarantees that all VMs requested are assigned to the servers. Constraint (8) makes sure that the number of instances allocated to VMs on server \( s_i \), \( 1 \leq i \leq M \), do not exceed the number of the same instance provided by that server. Constraints (9) and (10) determine the value of \( y_i \). Finally, the constraint specified in (11) indicates that decision variables \( x_{ijk} \) and \( y_i \) can only take 0 or 1.

Changing the objective function mentioned in Equation (2), we can change the goal of the optimization model. Keeping the same assumptions and constraints of Equation (2) and replacing \( F_1 \) with \( F_2 \), as shown in Equation (12), the proposed model optimizes the cost measure. Let \( C_{ik} \) represent the hourly price for running a VM of instance type \( I_k \) on server \( s_i \) so that the cost of a server is estimated as the sum of the costs of all instance types running on the server.

\[
F_2 : \sum_{i=1}^{M} \sum_{j=1}^{L} C_{ij} \sum_{k=1}^{N} x_{ijk}. \tag{12}
\]

To change the multiobjective optimization model to a single-objective one, we define a new function combining \( F_1 \) and \( F_2 \) as Equation (13), which assigns different weights to each of the functions and tries to optimize both of them simultaneously. The specified weights \( w_1 \) and \( w_2 \) determine the relative importance of each objective function. It turns out that changing \( w_1 \) and \( w_2 \) changes the final result of the model.

\[
F_3 = w_1 \times F_1 + w_2 \times F_2; \quad w_1 + w_2 = 1; \quad 0 \leq w_1.w_2 \leq 1. \tag{13}
\]

4 | ENERGY AND COST-AWARE VM CONSOLIDATION: PROPOSED ALGORITHM

Since the optimization model we are facing is NP-hard, finding a solution in this large-scale problem is time consuming. Therefore, we intend to reduce the search space of the problem using heuristic algorithms and to find an approximated optimal solution in shorter time. Hence, we propose a meta-heuristic algorithm based on genetic algorithm (GA) for solving the VM placement problem, ie, named energy and cost-aware virtual machine consolidation or “ECVMC”. The proposed method will accept the variables related to the VM placement problem as GA input parameters and provide a feasible VM placement plan as an output. As mentioned earlier, the placement plan shows that each VM selects which VM instance type should be run on which PM.

Figure 3 depicts a VM placement problem with 6 VMs, 3 instance types, and 4 servers. As shown in the figure, the optimized VM placement problem has 2 stages. First, each VM should select one instance type at a time. This selection can be based on the requirements of the application or software that VM runs on. We assumed that this selection is done randomly. Output results from this stage show the number of instance types selected by VMs. In the next stage, the appropriate PMs for hosting VMs will be selected. This selection is based on the objectives mentioned in the previous section.

In order to successfully apply the GA in the proposed algorithm, we should be defining accurate strategies and set appropriate parameters. Hence, we consider 5 basic steps of GA in the following sections.
4.1 Encoding schema

One of the most important elements of the GA is the chromosome. Representation or encoding of the chromosome affects the performance of the GA. There are different encoding schemes that have been applied to various problems. A chromosome features the feasible solution of our VM placement problem in cloud data centers. In this paper, each chromosome consists of 2 segments and the length of it is $2N + L + M - 2$. Since each VM has to be assigned to only one instance type on the first stage and just one VM has to be assigned to PM on the last stage of the problem, integer encoding was used to define this situation. In other words, each segment of the chromosome is a permutation of positive integers in a specified range. Therefore, the output result of chromosome encoding is a 1-dimensional array that facilitates the implementation of GA operators. The detailed structure of the chromosome is shown in Figure 3.

As shown in Figure 3, the first segment of the chromosome is comprised of $|N + L - 1|$ genes that is the permutation of integer numbers from 1 to $N + L - 1$. Numbers larger than $N$ are used as delimiters. Hence, the decoding schema of the first segment, from the beginning of the array to the first delimiter, indicates which of the VMs selected the first instance type for execution. From the first to the second delimiter, the VMs that have chosen the second instance type are specified. For example, the sequence (6,1,8,5,4,3,7,2) represents the value of genes in the first segment of the chromosome, which means that the number of allocated VMs with flavor instance type large is 2. These VMs are vm1 and vm6. Likewise, the VMs vm3, vm4, and vm5 and the VM vm2 choose medium and small instance types, respectively. Similarly, this work is repeated until the end of the segment. If 2 consecutive delimiters appear immediately, this means that none of the VMs select the specified instance type. As in the first segment, the length of the second segment is $|N + M - 1|$, which indicates that each VM chooses which PM to host. In order to decode this segment, the method presented in the previous segment is used.

4.2 Initialization step

In the first generation of the proposed algorithm, the initial population is randomly generated by taking into account the satisfaction of constraints (3) and (7) in Section 3.4. In order to reduce the computation time of GA execution, the range of values for each gene can be predetermined based on the encoding schema described in Section 4.1.
4.3 Evaluation function

Generally, one of the important issues in the GA is how to evaluate the performance of each chromosome in the population. Since each chromosome is proportional to a placement plan, the objective or fitness function (18) determines the power consumption and cost of placement plan based on the information received from the chromosome. The lower objective value implies the higher performance of the chromosome. Note that, if a certain chromosome violates any of constraints (4)-(6), penalize it according to penalty function (18). The amount of these penalties are calculated using

\[ v_{CPU}^i = \max \left( \frac{\sum_{j=1}^{L} \sum_{k=1}^{N} R_{j}^{CPU} x_{ijk}}{y_i \ast C_{CPU}^i} - 1.0 \right), \quad \forall i \in \{1, 2, \ldots, M\} \]  

(14)

\[ v_{Memory}^i = \max \left( \frac{\sum_{j=1}^{L} \sum_{k=1}^{N} R_{j}^{Memory} x_{ijk}}{y_i \ast C_{Memory}^i} - 1.0 \right), \quad \forall i \in \{1, 2, \ldots, M\} \]  

(15)

\[ v_{Disk}^i = \max \left( \frac{\sum_{j=1}^{L} \sum_{k=1}^{N} R_{j}^{Disk} x_{ijk}}{y_i \ast C_{Disk}^i} - 1.0 \right), \quad \forall i \in \{1, 2, \ldots, M\} \]  

(16)

\[ v_T = \frac{\sum_{i=1}^{M} \left( v_{CPU}^i + v_{Memory}^i + v_{Disk}^i \right)}{M}, \quad \forall i \in \{1, 2, \ldots, M\} \]  

(17)

\[ F = F_3 + (1 + \beta \ast v_T), \]  

(18)

where \( F_3 \) is obtained from Equation (13); and \( v_{CPU}^i, v_{Memory}^i, \) and \( v_{Disk}^i \) indicate the amount of penalties imposed for excessive use of the CPU, memory, and disk resource of \( i \)th PM, respectively. \( v_T \) is the average of all penalties and parameter \( \beta \) is a control parameter of the penalty function.

4.4 Selection strategy

There are various strategies for the selection of appropriate solutions in GA, such as roulette, rank biased, or uniform random. The goal of this strategy is to select one that favors well-fitted individuals and rejects the others. In the proposed model, we use the roulette-wheel selection strategy that assigns the probability of selecting each chromosome according to their fitness function value.

4.5 Crossover operator

The crossover operator in GA is used to produce new individuals from 2 parents. In each generation, the number of offspring that are added to the population by this operator is controlled by applying a parameter called crossover percentage (CP). We employed a segment-based crossover operator that was based on a single-point crossover. As shown in Figure 4, at first, a gene is selected randomly in each segment of the chromosome, and then the created fragments of each parent to produce offspring are combined together. The proposed crossover operator for GA is described in Algorithm 1.
Like the crossover operator, this operator is used to prevent early convergence and discover a new solution. However, unlike the crossover operator, this operator usually changes the values of a gene. In each generation, the number of offspring that are added to the population by this operator is determined using a parameter called mutation percentage (MP). As seen in Figure 5, in the proposed algorithm, a parent first chooses from the population randomly, then 2 genes of each segment are randomly selected. Finally, to generate a new individual, we exchange the gene value with each other. The proposed mutation operator for GA is described in Algorithm 2.
Algorithm 2: Mutation

1. **Input:** a chromosome, \( CH = a_1 a_2 \ldots a_q \)
2. **Output:** a mutated chromosome, \( CH' = a'_1 a'_2 \ldots a'_q \)
3. **For Each** segment in chromosome segments do
   4. \( CH' \leftarrow CH \)
   5. randomly generate two integer values between 1 and Segment Size called \( r1, r2 \);
   6. exchange the elements in a \( CH' \) at indexes \( r1 \) and \( r2 \);
7. **End For Each**
8. **return** \( CH' \)

### 4.7 ECVMC: the proposed algorithm

The overall pseudo-code procedure for solving the VM placement problem is outlined in Algorithm 3.

Algorithm 3: The proposed algorithm

1. **Input:** VMList, PMList, typeList
2. **Output:** allocation of VMs
3. Initially, \( Pop\ Size, CP, MP \) // number of population, Crossover Percentage, Mutation Percentage,
4. \( t \leftarrow 0 \); // Termination Condition;
5. \( nc \leftarrow Pop\ Size * CP \); // number of offspring;
6. \( nm \leftarrow Pop\ Size * MP \); // number of mutants;
7. \( population \leftarrow InitializePopulation(Pop\ Size) \); // Generate an initial random population
8. EvaluatePopulation(\( population \)); // Evaluates the population according Equation 18
9. **While** the termination condition not true do
10. \( t \leftarrow t + 1 \);
11. **For** \( i = 1..nc/2 \) // apply selection and crossover
12. \( parents \leftarrow Selection(population,2) \);
13. \( offsprings \leftarrow Crossover(parents) \);
14. EvaluatePopulation(\( offsprings \));
15. \( population.add(offsprings) \);
16. **End For**
17. **For** \( i = 1..nm \) // apply mutation
18. \( parent \leftarrow Selection(population,1) \);
19. \( offspring \leftarrow Mutation(parent) \);
20. EvaluatePopulation(\( offspring \));
21. \( population.add(offspring) \);
22. **End For**
23. \( population.sort() \); // Sort the individuals according to their fitness function
24. \( allocation \leftarrow population.get(\text{first}) \); // Select the best individual
25. **return** allocation;

### 5 PERFORMANCE EVALUATION

In this section, we present the performance of the proposed VM placement algorithm regarding the optimization criteria previously discussed (e.g., power and cost).

#### 5.1 Experimental setup

In all our experiments, we have used the system configuration comprised of a cloud provider with several servers. Each server is equipped with an Intel CoreTM i7-4710HQ 2.50 GHz processor, supporting 4 real cores and 4 virtual cores,
TABLE 2  Hardware configurations for virtual machine instance types

<table>
<thead>
<tr>
<th>Instance type</th>
<th>Small</th>
<th>Medium</th>
<th>Large</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU (# Cores)</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Memory (GB)</td>
<td>2</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>Disk (GB)</td>
<td>50</td>
<td>70</td>
<td>130</td>
</tr>
<tr>
<td>Instance type prices ($/h)</td>
<td>0.11</td>
<td>0.14</td>
<td>0.18</td>
</tr>
<tr>
<td>Power consumption (Watt/h)</td>
<td>37</td>
<td>27</td>
<td>22</td>
</tr>
</tbody>
</table>

TABLE 3  Algorithm parameters

<table>
<thead>
<tr>
<th>Genetic parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crossover percentage (CP)</td>
<td>80%</td>
</tr>
<tr>
<td>Population size (Pop Size)</td>
<td>100</td>
</tr>
<tr>
<td>Mutation percentage (MP)</td>
<td>10%</td>
</tr>
<tr>
<td>Selection type</td>
<td>Roulette-wheel</td>
</tr>
<tr>
<td>β</td>
<td>0.8</td>
</tr>
</tbody>
</table>

16GB of main memory, and 1TB of disk storage running on Ubuntu 16.04. Servers communicated via an interconnected network connecting any arbitrary pairs of servers to each other. The homogeneous servers are independent in which they can be switched on or off separately. To be able to run a set of independent VMs on each server, the KVM virtualization software is installed on each server.

As shown in Figure 2, a server can host at most 3 different VM instance types, i.e., small, medium, and large. The configuration of all VM instances and their related prices are listed in Table 2. In addition, the parameters used by the proposed algorithm are described in Table 3. The values of these parameters are not random and are based on literature and on the experience of conducting various experiments by setting different parameters. The proposed algorithm has been implemented in Java programming language. The underlying operation system is Linux Ubuntu 16.04.

5.2  Used algorithms

To demonstrate the effectiveness of the proposed algorithm, we compare the algorithm with 3 other VM placement algorithms, namely, first fit (FF), first fit decreasing (FFD), and permutation pack (PP). The FF algorithm deals with a list of VMs in given order at a time. It will allocate the given VM to the FF nonempty active server. If all active servers do not have enough resources to allocate to the current VM, a new server will be activated and the VM will be allocated. In the FFD algorithm, at first, all VMs are arranged according to a sorting criteria and then assign the VMs to the servers using the FF algorithm. In the VM allocation, the PP is an algorithm for the VMs with different resources with the goal of minimizing the wastage of resources and the number of required servers. In this algorithm, the VMs that need more resources are placed on the same server, whereas the other VMs are consolidated in the other set of servers. In this paper, each instance type of VM is consolidated in the same set of servers. In all experiments related to Sections 5.3.2 and 5.3.3, the VMs are sorted based on power consumption and cost, respectively.

5.3  Comparing conditions

In general, in order to evaluate the performance of the new algorithm, it should be compared with other algorithms. To achieve this, we need to perform various experiments. Hence, the proposed algorithm was tested in 3 parts. The first experiment is dedicated to the discussion of the characteristics of the proposed algorithm. The second experiment is related to the comparison of the power consumption of the proposed algorithm against 3 heuristic algorithms, i.e., FF, FFD, and PP. The third experiment focuses on comparing the placement plan cost of the proposed algorithm with existing heuristic algorithms.

The data set used to evaluate the performance evaluation of the algorithms in each experiment is similar to the data set described in the work of Hallawi et al., which is constructed using the following parameters:

\[ N = [10 - 350] \quad \text{number of VMs} \]

\[ M = 128 \quad \text{number of servers} \]
5.3.1 | Results

To evaluate the proposed algorithm, 420 experiments were carried out to analyze the performance. These experiments explore the behavior of the ECVMC algorithm in 3 situations. The experiments were performed using a problem of size 70 VMs over 10 runs. The output results are shown in Figure 6. In the first situation, the behavior of the algorithm for minimizing power consumption has been investigated. As shown in Figure 6A, with increasing algorithm parameters such as the number of generations, the performance of the derived solution is increased but requires more computational time. In each generation, the crossover and mutation operators add the new individuals to the population. This led to the elimination of individuals with the lowest fitness to the population and replacing them with better fitness individuals. Therefore, the performance of the algorithm performance improves.

Figure 6B shows the behavior of the algorithm trying to minimize the cost for different values of the number of the generation parameter. As shown in this figure, ECVMC easily converges after 40 generations. Finally, in Figure 6C, the simultaneous minimization of the power consumption and the cost of the proposed algorithm for different values of $w_1$ and $w_2$ is shown.

These experiments were performed using a problem instance discussed in (5.1). The problem sizes are varied by changing the number of VMs from 10 to 350.
TABLE 4  Results of the energy and cost-aware virtual machine consolidation (ECVMC) algorithm on different scenarios

<table>
<thead>
<tr>
<th>Scenario</th>
<th>#server</th>
<th>#VM</th>
<th>F1 (minimize power consumption)</th>
<th>F2 (minimize cost)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Opt ECVMC Gap%</td>
<td>Opt ECVMC Gap%</td>
</tr>
<tr>
<td>128</td>
<td>10</td>
<td>276</td>
<td>280 1.37 1.1</td>
<td>1.1 0</td>
</tr>
<tr>
<td>128</td>
<td>20</td>
<td>515</td>
<td>526 2.07 2.19</td>
<td>2.2 1</td>
</tr>
<tr>
<td>128</td>
<td>30</td>
<td>678</td>
<td>700 3.21 3.29</td>
<td>3.3 1</td>
</tr>
<tr>
<td>128</td>
<td>40</td>
<td>974</td>
<td>1004 3.10 4.39</td>
<td>4.4 1</td>
</tr>
<tr>
<td>128</td>
<td>50</td>
<td>1161</td>
<td>1206 3.86 5.49</td>
<td>5.5 1</td>
</tr>
<tr>
<td>128</td>
<td>60</td>
<td>1358</td>
<td>1402 3.22 6.59</td>
<td>6.6 1</td>
</tr>
<tr>
<td>128</td>
<td>70</td>
<td>1646</td>
<td>1711 3.94 N/A</td>
<td>7.7 N/A</td>
</tr>
<tr>
<td>128</td>
<td>80</td>
<td>1879</td>
<td>1950 3.79 N/A</td>
<td>8.8 N/A</td>
</tr>
<tr>
<td>128</td>
<td>90</td>
<td>2124</td>
<td>2209 3.98 N/A</td>
<td>9.9 N/A</td>
</tr>
<tr>
<td>128</td>
<td>100</td>
<td>2325</td>
<td>2408 3.59 N/A</td>
<td>10.1 N/A</td>
</tr>
<tr>
<td>128</td>
<td>120</td>
<td>2711</td>
<td>2808 3.56 N/A</td>
<td>12.2 N/A</td>
</tr>
<tr>
<td>128</td>
<td>140</td>
<td>3262</td>
<td>3359 2.96 N/A</td>
<td>14.4 N/A</td>
</tr>
<tr>
<td>128</td>
<td>160</td>
<td>3802</td>
<td>3956 4.05 N/A</td>
<td>16.6 N/A</td>
</tr>
<tr>
<td>128</td>
<td>180</td>
<td>4252</td>
<td>4405 3.60 N/A</td>
<td>18.8 N/A</td>
</tr>
<tr>
<td>128</td>
<td>200</td>
<td>4824</td>
<td>5003 3.70 N/A</td>
<td>22 N/A</td>
</tr>
<tr>
<td>128</td>
<td>225</td>
<td>N/A</td>
<td>5445 N/A 24.75 N/A</td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td>3.33 0.8</td>
<td></td>
</tr>
</tbody>
</table>

In order to compare the results from the ECVMC algorithm against the optimal result, the proposed model is solved by the CPLEX toolkit under Java. Table 4 shows a comparison of the results of the ECVMC algorithm with the results obtained by the solver for different scenarios. The first 2 columns define the scenario. More precisely, column 1 (#Server) gives the number of servers in the instance, whereas column 2 (#VM) shows the number of VMs. In the next columns, we show Opt, ECVMC, and Gap% values for the first objective function (F1) and second objective function (F2), respectively. Gap% is a relative error from optimum value, and it is calculated as $\text{Gap}\% = 100 \times \frac{\text{UB} - \text{LB}}{\text{LB}}$ (where UB and LB stand for the optimum value and the ECVMC value, respectively). A zero Gap% means that the solution found by the proposed algorithm is proven optimal, whereas an open gap means that there may exist a better solution in the range between UB and LB.

Observing Table 4, we notice that the solutions obtained for the proposed algorithm are near optimal because the gaps to optimal are far less than 4%. When considering large scenarios, as the size of the VMs grow, the solver will not be able to solve the model and obtain the optimal solution. In order to overcome this limitation, the proposed algorithm will be able to achieve a near optimal solution for larger scenarios.

5.3.2  Comparison of power consumption

To measure the power consumption of a server, we conducted several experiments to generate processor intensive workload by means of the SysBench benchmark suite. In all experiments, an instance type of a VM is executed on top of a server, then the workload is launched on the VM instance and the power consumption is measured by means of PowerStat. This procedure is done for all instances and the power consumed by each instance is measured. The input parameters of the proposed model with the F1 objective function (e.g., $P_{\text{idle}}$, $P_{\text{switch}}$, and $P_{\text{vm}}$) are obtained from the experiments. According to these experiments, the value of $P_{\text{idle}}$, $P_{\text{switch}}$, and $P_{\text{vm}}$ were equal to 15, 5, and 3, respectively.

Figure 7 shows the comparison of ECVMC, FF, FFD, and PP. It can be clearly observed that ECVMC produces better results compared to the other used algorithms. It has succeeded to reduce power consumption of the data center by 27% when compared to FFD. At the same time, the total power consumption of the data center is decreased by 35% and 24% compared to FF and PP. The FF shows the worst performance among the compared algorithms, which means that mapping the VMs on the servers based on the order of entry is inefficient for finding an optimal placement. In FFD, the performance of the algorithm has been improved by sorting out and prioritizing VM placement in terms of power consumption. The PP algorithm, due to the mapping of similar VMs on the same set of servers, has better results than the other 2 algorithms (FF and FFD). In ECVMC, in each generation, the crossover and mutation operators add the new
individuals to the population. After sorting population in terms of fitness, placement plans with the worst fitness were removed. At the end of each generation, the placement plan with the smallest fitness is chosen as a near-optimal solution. Figure 8 compares the average number of active servers at different algorithms. It shows that the ECVMC has improved the number of used servers at FF, FFD, and PP by 38%, 34%, and 29%, respectively. As can be seen, in all problem instances, the number of active servers, which were used for placement in the ECVMC, are less than the FF, FFD, and PP algorithms. This variance is due to multiple executions of the ECVMC algorithm and the improvement realized at each execution.

5.3.3 | Comparison of cost

The goal of the experiments that were performed in this section is to compare the new algorithm with its predecessor from literature, focusing on the cost. This comparison demonstrates the benefit gained by using the proposed algorithm against the conventional heuristic FF, FFD, and PP. As the number of VM instance types used in FF, FFD, and PP algorithms are the same, the difference between these three algorithms is the way in which VMs are assigned on different servers. As a result, the cost of the algorithms remain equal to one another. Hence, the comparison with an algorithm is done using the same data set described in Section 5.3.2.
Figure 9 demonstrates the performance of allocated VM requests in a data center by ECVMC and FFD (FF or PP) based on the cost of allocation. The cost is calculated by Equation (13) where coefficients w1 and w2 are assumed to be 0 and 1, respectively. It shows that the ECVMC has improved the placement cost of FFD by 75%. As noted in the previous section, this difference is because of the improvement obtained from each execution of the algorithm. Unlike the FFD algorithm, various algorithm characteristics such as initial population, mutation rate, crossover rate, and number of generations affect the outcomes of the ECVMC algorithm.

6 | CONCLUSION

In this paper, we considered a VM consolidation problem with the aim of minimizing power consumption and the cost of heterogeneous data centers. We provided a MINLP based formulation for the VM consolidation problem, and proposed a GA based meta-heuristic algorithm ECVMC for solving it. Finally, we compared its performance with three algorithms over a set of instances. It was found that ECVMC is superior to other tested algorithms, as it reduced the number of active servers by 38% over FF, 34% compared with FFD, and (29%) of PP. Minimizing the number of active servers leads to maximizing resource utilization and minimizing power consumption in total. The results show that ECVMC algorithm are capable of reducing power consumption of algorithms FF, FFD, and PP by 35%, 27%, and 24%, respectively. Furthermore, the algorithm reduced the cost of placement process by finding the near optimal solution. The future plan is considering a dynamic VM placement which needs to go through time-varying problem size.

ORCID

Amir Masoud Rahmani http://orcid.org/0000-0001-8641-6119

REFERENCE

37. IBM ILOG CPLEX. http://www.ilog.com/products/cplex

---

**How to cite this article:** Yousefipour A, Rahmani AM, Jahanshahi M. Energy and cost-aware virtual machine consolidation in cloud computing. *Softw Pract Exper.* 2018;1–17. https://doi.org/10.1002/spe.2585