A metaheuristic method for joint task scheduling and virtual machine placement in cloud data centers

ALBOANEEN, Dabiah, TIANFIELD, Huaglory, ZHANG, Yan and PRANGGONO, Bernardi <http://orcid.org/0000-0002-2992-697X>

A Metaheuristic Method for Joint Task Scheduling and Virtual Machine Placement in Cloud Data Centers

Dabiah Alboaneen[a,*], Huaglory Tianfield[b], Yan Zhang[b], Bernardi Pranggono[c]

[a] *Computer Science Department, College of Science and Humanities, Imam Abdulrahman Bin Faisal University, P.O.Box 31961, Jubail, Kingdom of Saudi Arabia*
[b] *Department of Computing, Glasgow Caledonian University, Glasgow, United Kingdom*
[c] *Department of Engineering and Mathematics, Sheffield Hallam University, Sheffield, United Kingdom*

Abstract

The virtual machine (VM) allocation problem is one of the main issues in the cloud data centers. This article proposes a new metaheuristic method to optimize joint task scheduling and VM placement in the cloud data center called JTSVMP. The JTSVMP problem composed of two parts, namely task scheduling and VM placement, is carried out by using metaheuristic optimization algorithms (MOAs). The proposed method aims to schedule task into the VM which has the least execution cost within deadline constraint and then place the selected VM on most utilized physical host (PH) within capacity constraint. To evaluate the performance of the proposed method, we compare the performance of task scheduling algorithms only with others that integrate both task scheduling and VM placement using MOAs, namely the basic glowworm swarm optimization (GSO), moth-flame glowworm swarm optimization (MFGSO) and genetic algorithm (GA). Simulation results show that optimizing joint task scheduling and VM placement algorithm leads to better overall results in terms of minimizing execution cost, makespan and degree of imbalance and maximizing PHs resource utilization.

Keywords: Cloud, Data center, Metaheuristic, Task scheduling, Virtual machine placement

1. Introduction

Cloud computing is a model for delivering on-demand computational services and resources such as computing power and data storage over the Internet [1]. Cloud computing provides resources as virtual machines (VMs) on-demand to users and executes their tasks in a way that meets quality of service (QoS)

*Corresponding author
Email address: dabuainain@iau.edu.sa (Dabiah Alboaneen)

requirements. Virtualization technology improves energy efficiency in data center by reducing the number of hardware in use and increase the utilization of resources by loading more than one virtual machine (VM) on a physical host (PH). Cloud provider need to schedule users' tasks into VMs and carefully place these VMs to physical hosts (PHs) in a way that considers both providers' and the users' optimization objectives.

Cloud computing utilizes data center to provide these services. It is predicted that by 2021, 94% of workloads will be managed by cloud data centers [2]. In a cloud data center, resources management can be done at two levels: (i) The first level is the task scheduling [3]; in this level each user's task is mapped onto suitable VM. When users' tasks need to be scheduled, users usually sign a service level agreement (SLA) with cloud provider. SLA is a contract between user and cloud provider on the expected service quality. In the SLA, the QoS requirements of the users should be clearly defined such as, the deadline of each task, budget, and service security level. Each cloud user has to decide which and how many VMs need to be provisioned before actually requesting and paying for the service. Hence, tasks scheduling directly affects the performance of cloud computing since inefficient tasks scheduling can lead to revenue loss, performance degradation and SLA violation. (ii) The second level is the VM placement [4]. VMs need to be placed in PHs that is capable of providing the required resources (i.e., processor, memory, and disk space). Therefore, the optimal VM placement plays an important role in improving resource utilization in a cloud computing environment.

The two levels are connected via VMs. Although VMs play an important role, we argue that VMs are just a tool for mapping users' tasks to PHs. The major users' aim is to find PHs for their tasks. On the other hand, providers' aim is to utilize their infrastructure by accommodating tasks for users. Therefore, task scheduling and VM placement problems influence each other.

Task scheduling in cloud computing can be modeled as a bin-packing problem and they are a non-deterministic polynomial-time hard (NP-hard) problem [5][6]. This problem becomes more challenging with the increase complexity of the cloud computing environment. Generally, it is difficult to develop algorithms to produce optimal solutions within a short time. Recently, using metaheuristic algorithms to deal with task scheduling and VM placement has received increasing attention due to the ability of the algorithms to provide near-optimal solutions within a reasonable time [7].

Previous works in this area, focused mostly on task scheduling such as [8] or VM placement such as [9] [10] [11] [12] as a separate problem. Both problems need to be addressed and integrated in order to produce an efficient solutions for both cloud users and providers.

In this article we investigate the research question:"to what extent joint task scheduling and VM placement can increase the performance in terms of execution cost, makespan, degree of imbalance and resource utilization?" To answer the research question, both task scheduling and VM placement are integrated and modeled as one optimization problem, called joint task scheduling and VM placement (JTSVMP). Metaheuristic optimization algorithms (MOA) are then used to solve this integrated problem and produce a schedule defining not only the task to VM mapping but also the VM to PH mapping. Integrating VM placement algorithm with task scheduling is more complicated under the task - VM - PH architecture, which causes three challenges:

(i) Which VM should be selected for a task?

(ii) Which PH should be selected for a VM?

(iii) How to simultaneously integrate the VM - PH placement to task - VM scheduling?

In summary, the key contributions of this article are as follows:

(i) Integration of task scheduling and VM placement as one optimization problem to produce better optimization of resource utilization in cloud data center. Specifically, the relationship between task, VM and PH is considered and the two-level architecture (i.e., task - VM and VM - PH) is extended to a three-level architecture (i.e., task - VM - PH).

(ii) Developing of MOA-based task scheduling under the three-level architecture (i.e., task - VM - PH). The proposed algorithm aims to simultaneously optimize the execution cost while meeting the deadline constraint when scheduling tasks to VMs, and to optimize the resource utilization of PHs when placing the VMs in PHs.

(iii) Performance evaluation of the proposed JTSVMP method through simulations. We compare the performance of task scheduling algorithms only with others that integrate both task scheduling and VM placement using MOAs, namely the basic glowworm swarm optimization (GSO), moth-flame glowworm swarm optimization (MFGSO) and genetic algorithm (GA). Simulation results show that optimizing joint task scheduling and VM placement leads to better overall results in terms of minimizing execution cost, makespan and degree of imbalance (DoI) and maximizing PHs resource utilization.

(iv) Statistical validation of the obtained results against that of GSO, MFGSO and GA using significance test. We use Wilcoxons rank-sum test.

The remainder of this article is arranged as follows. Section 2 presents the related work on tasks scheduling and VM placement in cloud data center using metaheuristic algorithms. Section 3 describes scheduling models and problem formulation. The proposed MOA is presented in Section 4. Section 5 presents the experimental evaluation. Finally, Section 6 draws the conclusion and future work.

Table 1: Existing metaheuristic algorithms for task scheduling and VM placement

Reference	Task scheduling	VM placement	Optimization algorithm
[13] [14] [15] [16]	✓		GA
[17] [18] [19]	✓		ACO
[20]	✓		PSO
[21] [22]	✓		Hybrid of PSO & SA
[23]	✓		Stochastic hill climbing
[24]	✓		ABC, PSO, ACO
[25]	✓		Hybrid of GA & ACO
[26]	✓		SOS
[27] [28] [29] [30] [31] [30]		✓	GA
[32] [33]		✓	SA
[34] [35]		✓	BBO
[36] [37] [38] [39]		✓	PSO
[40] [41] [42] [43] [44] [45] [46]		✓	ACO

2. Related Work

In this section, related studies on task scheduling and VM placement in cloud computing using metaheuristic algorithms are presented. A significant amount of research has focused on task scheduling and VM placement as shown in Table 1.

2.1. Metaheuristic Algorithms for Task Scheduling in Cloud Computing

Task scheduling based on GA has been studied widely in [13][14][15][16]. Zhu *et al.* use hybrid GA algorithm to solve only load balancing when scheduling tasks in cloud computing [16].

Task scheduling based on ant colony optimization (ACO) algorithm for load balancing and minimizing the average execution time is studied in [17]. Simulation results showed that the proposed algorithm outperformed first come first serve (FCFS) and the basic ACO algorithms.

A similar study by Tawfeek *et al.* also use ACO to minimize the execution time of tasks and the simulation results showed that the ACO outperformed FCFS and round robin (RR) algorithms [18].

ACO is improved to get a better performance when scheduling tasks in the cloud computing. The simulation results showed that the proposed algorithm had a good performance in minimizing the execution time and balancing the load [19].

Task scheduling in view of both the task execution time and the system resource utilization based on an improved particle swarm optimization (PSO) algorithm is proposed in [20].

In [21] and [22], a hybrid of PSO and simulated annealing (SA) is implemented on CloudSim to schedule tasks in the cloud. The results showed that the proposed algorithms can reduce the average execution time of task and increase resource utilization.

In [23], a stochastic hill climbing algorithm is used to schedule tasks to VMs. Simulation results based on CloudAnalyst simulator showed the efficiency of the proposed algorithm when compared to RR and FCFS algorithms.

In [24], three different metaheuristics approaches (i.e., artificial bee colony (ABC), PSO and ACO) have been evaluated for cloud task scheduling. The proposed algorithms are better in minimizing the total execution time compared to LTF, random and FCFS algorithms. Moreover, ABC algorithm outperformed other algorithms. The PSO and ACO came in second level and third level, respectively.

Discrete version of Symbiotic Organism Search (SOS) algorithm for optimal scheduling of tasks on cloud resources called DSOS is proposed in [26]. Simulation results revealed that DSOS outperforms PSO for task scheduling problems particularly for large search space.

Moreover, integrating ACO algorithm with GA for scheduling tasks is proposed by Dai *et al.* This algorithm considered multiple QoS constraints in the scheduling process and it has superior performance in balancing resources and minimizing execution time [25].

However, these algorithms mainly focused on improving the execution time and resource utilization when scheduling tasks to VMs. Moreover, the execution time did not include the waiting time of tasks while in our algorithm we consider the execution time, waiting time, and execution cost of tasks.

2.2. Metaheuristic Algorithms for VM placement in Cloud Computing

In metaheuristics approaches, different algorithms have been proposed for optimizing VM placement in cloud computing. One of the main objectives considered by most existing researches is the energy consumption. GA [27], SA [32] [33], biogeography-based optimization (BBO) [34], PSO [36] [37] and ACO [40] [41] are used for energy-efficient VM placement. On the other hand, some researches focused on maximizing the performance rather than minimizing the energy [47].

Moreover, the trade-off between minimizing the energy and maximizing the performance is an important issue and needs to be addressed when formulating the VM placement problem. GA [28] [29] [30] [31], SA [48], PSO [38] [39], ACO [42] [43] [44] [45] [46] and BBO [35] are used to solve VM placement problem for energy and performance purposes.

2.3. Motivation

One of the identified gaps associated with existing studies in optimizing resource scheduling in cloud computing is that existing research works consider managing and scheduling resources in data center at two different levels separately: (i) task scheduling; (ii) VMs placement. Therefore, an optimal VMs placement plays an important role in improving resource utilization in a cloud computing environment.

However, researchers often addressed and evaluated the two levels individually by developing two-level architecture for optimizing tasks scheduling only for cloud users' benefit or for optimizing VMs placement for cloud providers' benefit.

The major users' aim is to find PHs for their tasks and the cloud providers' aim is to utilize their infrastructure by accommodating tasks for users. Therefore, task scheduling and VM placement problems effect each other.

The proposed three-level architecture addresses the identified research gap by developing a generalized architecture for simultaneously optimizing the two levels, i.e., tasks scheduling and VMs placement to obtain a better results for both cloud users and cloud providers. MFGSO [49] algorithm is applied to optimize JTSVMP under the three-level architecture (i.e., task-VM-PH).

In our earlier works, GSO algorithm has been applied for dynamic VM placement only [50] and for task scheduling only [51]. In this article, the integration of task scheduling and VM placement problems, as one optimization problem is proposed in order to produce better optimization of resource scheduling in cloud data center. Hence, the relationship between task, VM, and PH is considered and the existing two-level architecture (task - VM and VM - PH) is extended to a three-level architecture (task - VM - PH).

The rationale behind our proposed architecture is to schedule tasks to least executed cost VMs in such a way that tasks sequentially executed in the VM complete before their deadlines. In addition, placing the VMs on most utilized PHs to reduce the number of active PHs.

3. System Model and Problem Formulation

In this section, the architecture of JTSVMP in cloud-based data center is described. Mathematical models that represent the scheduling, cost and resource utilization models are presented. The problem of scheduling tasks to VMs with considering placing VMs to PHs given the constraints is considered as well.

Figure 1 depicts the architecture of JTSVMP in cloud-based data center. The architecture consists of three layers: users layer, scheduling layer, and computing resource layer. In the users layer, users dynamically submit their tasks to cloud providers in a given time interval.

The targeted system is a large-scale data center. The computing resource layer consists of heterogeneous PHs where each PH hosts a set of heterogeneous VMs via the corresponding virtual machine monitor (VMM). Having heterogeneous VMs with varied processing speeds and memory, indicating that a task executed on different VMs will lead in varying execution cost. Each PH has a *local monitor*, a software module, which is responsible for collecting -from VMM- run-time statistics of each PH, including PH status and resource utilization of all VMs in a PH and reports them to the *task scheduler*.

The scheduling layer consists of a *task scheduler* and *VM placement algorithm*. *Task scheduler* is responsible for scheduling tasks to VMs and then the *VM placement algorithm* is responsible for placing selected VMs to PHs.

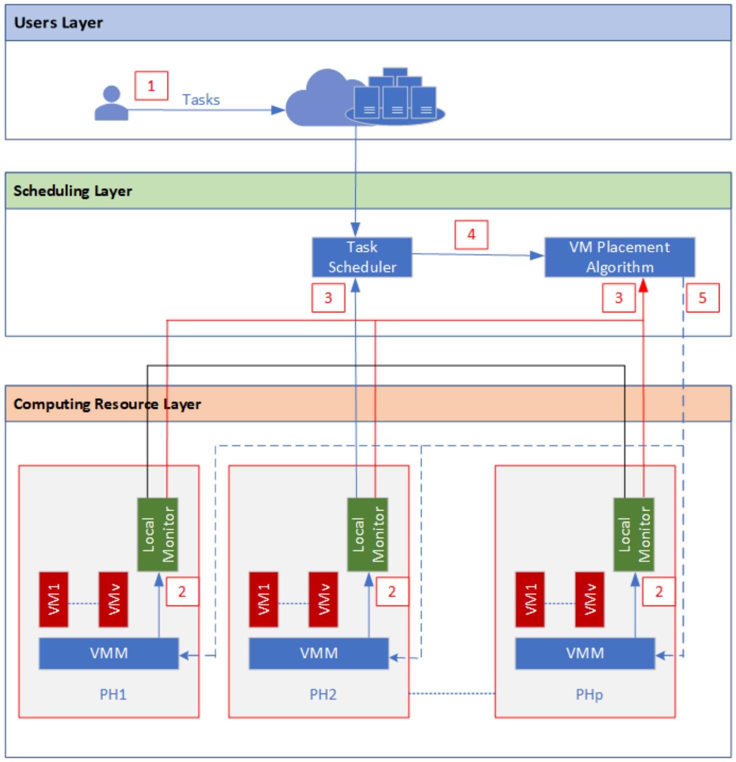


Figure 1: Task scheduling and VM placement

Our aim is to schedule tasks to VMs in order to achieve lower execution cost of tasks while meeting the deadline of tasks. As a result, the deadline is equal to the expected execution time of the tasks to be scheduled on each VM. The *task scheduler* first calculates the completion time required to execute the task on each VM which is based on the execution time and waiting time of task. The execution time is calculated based on the ratio of the length of task which is received from the user side as a number of instructions and the processing speed of VM which is received from the *local monitor* side in terms of million instructions per second (MIPS). If the completion time of executing task is within the task deadline, then the task can be executed on the VM. After that, the *task scheduler* calculates the execution cost of task in each available VM. To minimize the execution cost of the task, *task scheduler* decides which VM has the least execution cost for executing the task and meets the requirements of each task.

The *VM placement algorithm* is responsible of placing the selected VM on PH. Here, the *VM placement algorithm* calculates the CPU utilization of each PH by considering the total and the available CPU and then allocates the VM on the most utilized PH in order to maximize the resource utilization. Finally,

Table 2: Amazon EC2 instance types and prices

VM type	MIPS	Pe	Capacity	Price ($/Hour)
Type 1 (sml)	500	4	2000	$0.34
Type 2 (med)	1000	7	7000	$0.5
Type 3 (lrg)	1500	20	30000	$0.6

the *task scheduler* will return the result of execution to user when all tasks are completed.

3.1. Tasks Model

Set of tasks is defined as $T = \{T_1, T_2, ..., T_i, ..., T_M\}$, where $i \in [1, M]$ and M is the total number of tasks. Each task T_i is described as $T_i(sz_i, de_i, a_i)$, where sz_i, de_i and a_i represent the task size that is measured by million of instructions (MI), task deadline and start time of task T_i, respectively.

3.2. Virtual Machines Model

Set of VMs is defined as $VM = \{vm_1, vm_2, ..., vm_j, ..., vm_V\}$, where $j \in [1, V]$ and V is the total number of VMs. Each vm_j is described as $vm_j(cv_j, Price_j)$, where cv_j is the VM processing capacity which is expressed in terms of million instructions per second (MIPS) that is subject to $\sum_{j=1}^{V} cv_j \leq cp_k$. This information is used in the proposed algorithm to calculate the execution time of a task on a given VM. $Price_j$ is the amount of payment spent for using a vm_j per hour. We consider three types of VMs offered by the cloud provider to the user as $\{vm_{sml}, vm_{med}, vm_{lrg}\}$, and each VM type has different processing capacity and price. (Table 2 shows Amazon EC2 pricing model[1]).

3.3. Physical Hosts Model

Set of PHs is defined as $PH = \{ph_1, ph_2, ..., ph_k, ..., ph_P\}$, where $k \in [1, P]$ and P is the total number of PHs. Each ph_k is described as $ph_k(cp_k)$, where cp_k is the PH processing capacity which is expressed in terms of MIPS.

3.4. Cost and Time Models

We assumed that the cost of each VM within a cloud is dynamically affected based on the VMs performance type, which means a more powerful VM is always more costly.

Cost indicates the total amount the user needs to pay to cloud provider for renting the VMs. Minimizing the cost is one of the optimization parameters for user favor. It should be considered when formulating the task scheduling problem.

The unit of time in which the pay-per-use model is specified by the cloud provider; any partial utilization of the leased VM is charged as if the full-time period was consumed. For instance, if unit of time is 60 minutes, when a VM is used for 61 minutes the user will have to pay for 120 minutes.

A widely used model to calculate the cost is based on the execution time of task and the cost of VM per unit of time as used in [52] and [50]. Therefore, The execution

[1] http://aws.amazon.com/ec2/instance-types/

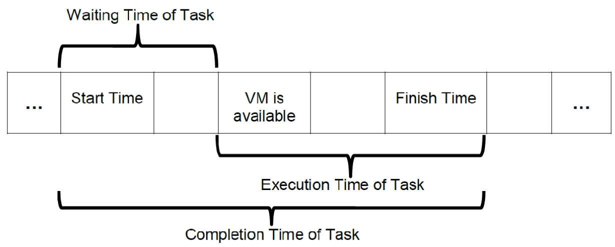


Figure 2: Task scheduling lifetime

cost EC_{ij} of T_i is defined as multiplication of the price of vm_j and the completion time of T_i, that is,

$$EC_{ij} = Price_j * \frac{CT_{ij}}{3600} \tag{1}$$

where $Price_j$ is the price of vm_j and CT_{ij} is the completion time of executing task T_i on vm_j.

If the completion time CT_{ij} is within the deadline, T_i can be executed. Otherwise, T_i cannot be executed. As seen in Figure 2, the execution time is the time that VM takes to execute the task. Waiting time is the time difference between task start time and task execution time. Hence, the completion time that vm_j will take to execute T_i can be calculated as Eq.2.

$$CT_{ij} = a_i + ET_{ij} \tag{2}$$

where a_i is the start time of T_i and ET_{ij} is the time of executing T_i on vm_j at a given time t.

A widely used model to estimate the execution time is based on the task size and the processing speed of VMs. The execution time ET_{ij} is calculated as Eq.3.

$$ET_{ij} = \frac{sz_i}{cv_j} \tag{3}$$

where sz_i is the number of instructions that T_i will need to execute on vm_j and cv_j is the processing speed of vm_j, which can be calculated as Eq.4.

$$cv_j = (Pe_j * mips_j) \tag{4}$$

where Pe_j is the number of processors in vm_j, $mips_j$ is million instructions per second of each processor in vm_j.

3.5. Resource Utilization Model

The resource (i.e., CPU) utilization u of ph_k for a given time t is calculated by Eq.5.

$$u_k(t) = \frac{\sum_{j=1}^{V} vp_{jk} * cv_j}{cp_k} \tag{5}$$

where vp_{jk} is a binary variable indicating whether vm_j is assigned to ph_k or not. The value of $vp_{jk} = 1$ if vm_j is assigned to ph_k, otherwise it is 0. cv_j and cp_k are the processing speed of vm_j and ph_k, respectively.

Table 3: Notations

Notation	Definition
i	Index for tasks
M	Total number of tasks
j	Index for VMs
V	Total number of VMs
k	Index for PHs
P	Total number of PHs
sz_i	Size of T_i
de_i	Deadline of T_i
a_i	Start time of T_i
EC_{ij}	Execution cost of executing T_i on vm_j
Tv_{ij}	Variable indicates whether T_i is assigned to vm_j
CT_{ij}	Completion time of executing T_i on vm_j
$Tcpu_i$	CPU demand of T_i
$Tmem_i$	Memory demand of T_i
$Tnet_i$	Network bandwidth demand of T_i
cv_j	Capacity of vm_j
Pe_j	Total number of processors in vm_j
$mips_j$	MIPS of each processor in vm_j
$Price_j$	Price of vm_j
$Vcpu_j$	CPU demand of vm_j
$Vmem_j$	Memory demand of vm_j
$Vnet_j$	Network bandwidth demand of vm_j
$selectedVm$	Index for the selected VM
cp_k	Capacity of ph_k
u_k	CPU utilization of ph_k
vp_{jk}	Variable indicates whether vm_j is assigned to ph_k
$Pcpu_k$	CPU capacity of ph_k
$Pmem_k$	Memory capacity of ph_k
$Pnet_k$	Network bandwidth capacity of ph_k

3.6. Problem Formulation

In this article, we consider scheduling independent tasks in cloud based data center comprising heterogeneous VMs and PHs. In this section, we introduce objective function and constraints considered in the problem.

The objective is to determine a plan for task scheduling and VM placement in order to minimize both the execution cost EC_{ij} of tasks and the available $MIPS$ of PHs AM_{jk} as below:

$$\min f = ECAM = (EC_{ij}(t) * 0.5) + (AM_{jk}(t) * 0.5) \tag{6}$$

Task Scheduling Constraints

(i) A task must be assigned to one VM, i.e.,

$$\forall i \in \{1, 2, ..., M\}, \sum_{j=1}^{V} Tv_{ij}(t) = 1 \tag{7}$$

where i is index of task, j is index of VM and $Tv_{ij}(t)$ is a binary value representing whether T_i is assigned to vm_j at given time t.

(ii) Ensures that each task is finished before its deadline, i.e.,

$$\forall i \in \{1, 2, ..., M\}, CT_{ij} \leq de_i \tag{8}$$

where de_i is the deadline of task T_i and CT_{ij} is the completion time of executing task T_i on vm_j.

(iii) The total requirements resources of all tasks hosted on VM should not exceed the maximum capacity of the VM resources, i.e.,

$$\forall j \in \{1, 2, ..., V\}, \sum_{i=1}^{M} Tcpu_i * Tv_{ij} \leq Vcpu_j \tag{9}$$

$$\forall j \in \{1, 2, ..., V\}, \sum_{i=1}^{M} Tmem_i * Tv_{ij} \leq Vmem_j \tag{10}$$

$$\forall j \in \{1, 2, ..., V\}, \sum_{i=1}^{M} Tnet_i * Tv_{ij} \leq Vnet_j \tag{11}$$

where $Tcpu_i$, $Tmem_i$ and $Tnet_i$ are CPU, memory and network bandwidth demands of T_i, respectively. $Vcpu_j$, $Vmem_j$ and $Vnet_j$ are CPU, memory and network bandwidth capacities of vm_j , respectively.

(iv) Tasks are real-time and independent of each other.

(v) All the tasks of CPU intensive.

(vi) Each task is allowed to be processed on any given available VM that meets the requirements of tasks.

(vii) The execution time of each task is VM-dependent.

(viii) Each task must be completed without interruption once started (non-preemptable). So, if more than one task comes at the same time, then one task will wait in the queue until previously task completed its execution.

(ix) Each VM can be provisioned to more than one task.

VM Placement Constraints

(i) A VM must be assigned to one PH, i.e.,

$$\forall j \in \{1, 2, ..., V\}, \sum_{k=1}^{P} vp_{jk}(t) = 1 \tag{12}$$

where j is index of VM, k is index of PH and $vp_{jk}(t)$ is a binary value representing whether vm_j is assigned to ph_k at given time t.

(ii) The total resources of a VM cannot exceed the capacity of the PH resources, i.e.,

$$\forall k \in \{1, 2, ..., P\}, \sum_{j=1}^{V} Vcpu_j * vp_{jk}(t) \leq Pcpu_k \tag{13}$$

$$\forall k \in \{1, 2, ..., P\}, \sum_{j=1}^{V} Vmem_j * vp_{jk}(t) \leq Pmem_k \tag{14}$$

$$\forall k \in \{1, 2, ..., P\}, \sum_{j=1}^{V} Vnet_j * vp_{jk}(t) \leq Pnet_k \tag{15}$$

where $Vcpu_j$, $Vmem_j$ and $Vnet_j$ are CPU, memory and network bandwidth demands of vm_j, respectively. $Pcpu_k$, $Pmem_k$ and $Pnet_k$ are CPU, memory and network bandwidth capacities of ph_k , respectively.

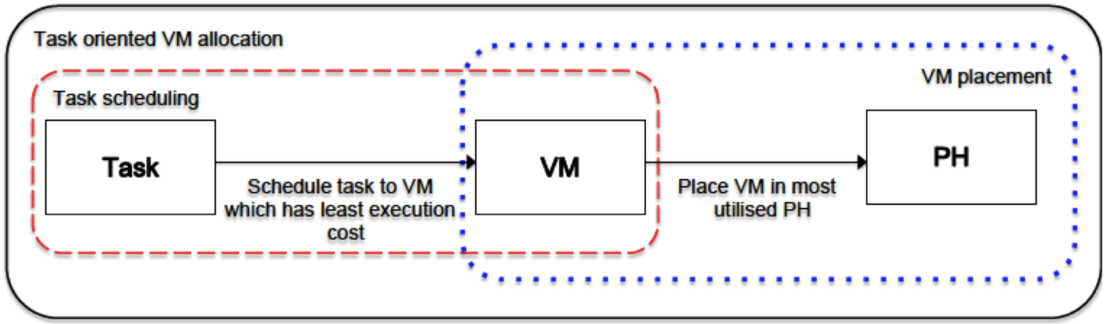


Figure 3: Joint task scheduling and VM placement

4. Solving JTSVMP by Metaheuristic optimization Algorithm

This section presents the proposed metaheuristic method for JTSVMP, composed of task scheduling and VM placement problems in cloud data centers.

The pseudo code of the two level architecture, task scheduling based-MOA, is shown in Algorithm 1. The proposed architecture integrates two levels namely task scheduling (level 1) and VM placement (level 2) by using MOAs. Figure 3 depicts the working principle of JTSVMP. The pseudocode of the MOA-based JTSVMP is shown in Algorithm 5.

The input parameters of the MOA-based JTSVMP include $vmList$, $taskList$ and $phList$ details. Each individual of MOA represents a VM and the location of VM is the execution cost; thus, the dimension of the individual is equal to the number of VMs. MOA parameters are initialized (line 1). When task i needs to be processed, a VM vm_j is randomly initialized (lines $3-4$). Then, for each task i that needs to be scheduled (line 5), on each VM vm_j in the $vmList$ (line 7), calculate the fitness of vm_j as per its execution cost (line 8). Then, the movement will be updated (line 9). The best suited vm $selectedVm$ is selected to schedule the task (line 10). Furthermore, the $selectedVm$ will be placed to a most utilized PH ph_k (line 11) and then $ECAM$ is calculated (line 12). Finally, the algorithm will be terminated if there is no improvement in reducing the $ECAM$ from the last iteration (line 15).

Metaheuristic algorithms which are applied to solve JTSVMP are, GSO and the hybrid GSO with MFO which is called, MFGSO. In GSO, the initial population is randomly generated, while in MFGSO, MFO is integrated to initialize the initial population of GSO instead of randomization. It is worth to mention that MFGSO has better performance than GSO in terms of reducing of being trapped into local optima and to speed up the convergence.

GSO algorithm is applied to search for VM that minimizes the execution cost. Each glowworm represents a VM and the luciferin of VM is the execution cost. According to the nature of glowworms, they always move towards their neighbors having higher luciferin than its own. But in our algorithm, a VM is attracted towards its neighbor which has lowest execution cost, which is reverse of the characteristics of the glowworm.

Task Scheduling based-GSO

The movement function of GSO algorithm is presented in Algorithm 2. The luciferin $\ell_j(t)$ (i.e., execution cost) of vm_j will be updated (line 1 in Algorithm 2). The neighbor set $N_j(t)$ will be calculated through $getVmNeighbours(vm_j, n_t)$ (line 2 in Algorithm 2), it contains VMs which have a lower execution cost than the original one and can meet the deadline of executing the T_i. The size of the neighbor set n_t is predefined by the user.

The luciferin of each other VM $\ell_n(t)$ will be calculated (line 3 in Algorithm 3). If a VM vm_n has less execution cost than the original VM vm_j and if it can meet the deadline of the task de_i, then this VM vm_n will be added to the neighbor list N_j as a neighbour of VM vm_j (lines $4-6$ in Algorithm 3).

The neighbor which has highest probability among neighbors n_{*j} will be selected to schedule the task (lines $3-5$) and the location of VM vm_j is updated (line 6). After that, the radial range $radius_j$ which defines the neighbor set will be updated (line 7).

Task Scheduling based-MFGSO

The movement function of MFGSO algorithm is presented in Algorithm 4. In MFGSO, MFO algorithm is used to initialize the initial population of GSO. The moths are considered as the candidate VMs. Flames are the best positions of VMs that are obtained so far by the VM. The number of flames N_F will be decreased in each iteration (line 1). The fitness values of each moth and flame are obtained (lines $2-4$). Flames are sorted based on its fitness values (execution cost) and saved in F in case of first iteration. However, in next iteration, F is the sorted of merge moths and best flames from previous iteration (line 5). Next, positions of moths are updated (lines $7-15$). The moth's current optimal position, $M_j(t+1)$ is obtained (line 15). Then, the current optimal position obtained from MFO, $vm_j(t)$ and corresponding fitness value $fit(vm_j(t))$ as the initial values of GSO algorithm are set. GSO is used to exploit the best solutions (lines $16-24$).

Algorithm 1: Task scheduling based-MOA

Input: $vmList$, $taskList$
Output: $selectedVm$

Set parameters of MOA
Set $t = 1$
for $T_i \in taskList$ **do**
    Initialize vm_j randomly
for $T_i \in taskList$ **do**
    while $termination\_condition\_not\_met$ **do**
        for $vm_j \in vmList$ **do**
            $fit(vm_j(t)) = Price_j * CT_{ij}$
            $vm_j(t+1) = mh\_movement(fit(vm_j(t)), vm_j(t))$
            $selectedVm = vm_j(t+1)$
        $t = t + 1$
    $taskList = taskList - i$
return $selectedVm$

Algorithm 2: Function $mh\_movement(fit(vm_j(t)), vm_j(t))$ in GSO algorithm

$\ell_j(t) = (1-\lambda)\ell_j(t-1) + \gamma fit(vm_j(t))$
$N_j(t) = getVmNeighbours(vm_j, n_t)$
for $n \epsilon N_j$ **do**
    $p_{jn}(t) = \frac{\ell_n(t)-\ell_j(t)}{\sum_{n\epsilon N_j(t)} \ell_n(t)-\ell_j(t)}$
$n_{*j} = \arg\max_{n\epsilon N_j(t)}\{p_{jn}(t)\}$
$vm_j(t+1) = vm_j(t) + \zeta\left(\frac{vm_{n_{*j}}(t)-vm_j(t)}{||vm_{n_{*j}}(t)-vm_j(t)||}\right)$
$radius_j(t+1) = \min\{max\_radius, \max\{0, radius_j(t) + \beta(max\#\_neighbour - |N_j(t)|)\}\}$

Algorithm 3: Function $getVmNeighbours(vm_j, n_t)$

$N_j \rightarrow NULL$
while $n \leq n_t$ **do**
    $\ell_n(t) = Price_n * CT_{in}$
    if $\ell_n(t) < \ell_j(t)$ **then**
        if $||CT_{in}(t) - CT_{ij}(t)|| \leq de_i$ **then**
            $N_j \leftarrow n$

Algorithm 4: Function $mh\_movement(fit(vm_j(t)), vm_j(t))$ in MFGSO algorithm

$N_F = round(V - t * \frac{V-1}{ITER})$
for $j = 1 : V$ **do**
    Evaluate $fit(moth^j)$
    Evaluate $fit(flame^j)$
$F(t) = sort(F(t-1), M(t))$ with the flames and moths fitness values from best to worst
$\phi = -1 + t * ((-1)/T)$
for $j = 1{:}V$ **do**
    for $s = 1 : N_F$ **do**
        $\nu = (\phi - 1) * rand + 1$
        $D_j(t) = |F_j(t) - M_j(t)|$
        if $j \leq N_F$ **then**
            $M_j(t+1) = S(M_j(t), F_j(t)) = D_j(t) * e^{b\nu}.\cos(2\pi\nu) + F_j(t)$
        if $j > N_F$ **then**
            $M_j(t+1) = S(M_j(t), F_{N_F}(t)) = D_j(t) * e^{b\nu}.\cos(2\pi\nu) + F_{N_F}(t)$
    $vm_j(t) = M_j(t+1)$
for $j = 1{:}\ V$ **do**
    $\ell_j(t) = (1-\lambda)\ell_j(t-1) + \gamma fit(vm_j(t))$
for $j = 1{:}\ V$ **do**
    for *each* $n \epsilon N_j(t)$ **do**
        $p_{jn}(t) = \frac{\ell_n(t) - \ell_j(t)}{\sum_{n \epsilon N_j(t)} \ell_n(t) - \ell_j(t)}$
    $n_{*j} = \arg\max_{n \epsilon N_j(t)} \{p_{jn}(t)\}$
    $vm_j(t+1) = vm_j(t) + \zeta \left(\frac{vm_{n_{*j}}(t) - vm_j(t)}{||vm_{n_{*j}}(t) - vm_j(t)||} \right)$
    $radius_j(t+1) = \min\{max\_radius, \max\{0, radius_j(t) + \beta(max\#\_neighbour - |N_j(t)|)\}\}$
    $M_j(t+1) = vm_j(t+1)$

Algorithm 5: MOA-based JTSVMP

Input: $vmList$, $taskList$, $phList$
Output: $selectedVm$

Set parameters of MOA
Set $t = 1$
for $T_i \in taskList$ **do**
    Initialise vm_j randomly
for $T_i \in taskList$ **do**
    while $termination\_condition\_not\_met$ **do**
        for $vm_j \in vmList$ **do**
            $fit(vm_j(t)) = Price_j * CT_{ij}$
            $vm_j(t+1) = mh\_movement(fit(vm_j(t)), vm_j(t))$
            $selectedVm = vm_j(t+1)$
            Place $selectedVm$ to ph_k
            $ECAM = (EC_{ij} * 0.5) + (AM_{jk} * 0.5)$
        $t = t + 1$
    $taskList = taskList - i$
return $selectedVm$

Table 4: Settings of data center

Parameter	RAM	Storage	BW	VM scheduler	VMM
Value	2 GB	1 TB	10 GB	Time-shared	Xen

Table 5: Settings of PHs

PH	Processor	Pe	MIPS
PH-A	Intel Core 2 Extreme X6800	2	27079
PH-B	Intel Core i7 Extreme 3960X	6	177730

Table 6: Parameter settings of the algorithms

Algorithm	Notation	Description of the parameter	Value
-	N	Number of population	50
-	R	Number of experimental runs	10
GSO	λ	Luciferin decay coefficient	0.4
	γ	Luciferin enhancement coefficient	0.6
	β	Rate of the neighbourhood range	0.08
	$max\#\_neighbour$	No. of neighbours	5
	max_radius	Maximum range	8
	ζ	Step size of moving	0.03
	ℓ	Initial luciferin	0.05
MFGSO	b	Constant defining the shape of the logarithmic spiral	1
GA	p_c	Crossover probability	0.8
	p_m	Mutation probability	0.2
	-	Selection mechanism	Roulette Wheel

5. Experimental Evaluation

5.1. Simulation Setup

CloudSim 3.0.3 toolkit [53] is used to evaluate the proposed architecture. CloudSim is widely used to simulate cloud system components such as data centers, tasks and VMs. It supports policies for tasks scheduling, VMs placement and selection, power models for data center resources and provides different types of workloads. We modeled an Infrastructure-as-a-Service (IaaS) provider offering a single data center, eight PHs with two different types and four VMs with three different types based on current Amazon EC2 offerings as shown in Table 2. The characteristics of data center and PHs are shown in Tables 4 and 5, respectively. Tasks are generated randomly and from a standard formatted workload of a NASA Ames Research center [54]. NASA Ames iPSC/860 set logs is one of the widely used formatted workloads for evaluating the performance of distributed systems [54] [55]. NASA Ames iPSC/860 set log contains information of 14,794 tasks. Different sizes of task are used started with 100 tasks to 500 tasks.

The algorithms that are compared include the basic GSO, MFGSO and GA in two different scenarios, when considering tasks scheduling only and when integrating VM placement with task scheduling, JTSVMP. In the experiments, we follow the recommended value of parameters for GSO and GA algorithms as presented in Table 6 [56] [57].

5.2. Simulation Results and Discussion

The proposed architecture of integrating task scheduling and VM placement are evaluated in this section. The evaluation considers two workloads: random and real workloads. Two scenarios is considered: the first scenario is task scheduling only and the second scenario is the integration of task scheduling and VM placement. The performance metrics are: execution cost, makespan, DoI and the resource utilization of PHs.

Table 7: Execution cost (EC)

Workload	No. of tasks	GSO	MFGSO	GA	IGSO	IMFGSO	IGA
Random	100	9.757	6.231	5.023	7.426	3.4	3.01
	200	18.480	15.26	10.183	14.912	12.10	8.37
	300	27.203	22.75	16.518	22.318	18.44	12.26
	400	35.950	29.35	19.170	29.769	24.22	15.9
	500	44.769	36.68	24.264	37.193	30.11	17.56
NASA Ames iPSC/860	100	219.850	178.01	87.015	184.789	165.43	56.33
	200	438.878	388.65	315.158	407.039	364.29	243.27
	300	745.878	512.55	409.220	526.302	418.07	298.46
	400	837.146	533.99	558.309	543.468	441.65	464.02
	500	1063.140	589.04	586.473	606.234	500.30	503.80

Execution Cost

Table 7 presents the results of execution cost in the random and NASA Ames iPSC/860 workloads for GSO, MFGSO, GA, integration GSO (IGSO), integration MFGSO (IMFGSO) and integration GA (IGA) algorithms.

According to the type of the VM used to run a task and the time required to complete the task, the execution cost of task can be calculated using Eq.1. Assuming the number of VMs is fixed as 4 VMs and the number of tasks is gradually increased from 100 to 500 tasks.

It can be seen from Table 7 that MFGSO outperforms the basic GSO algorithm in both workloads when generating different number of tasks. Moreover, GA has the best results in both workloads. Regards the effect of JTSVMP, in both workload IGSO, IMFGSO and IGA have less execution cost than the GSO, MFGSO and GA, respectively, which means the JTSVMP leads to improve the performance in terms of minimizing the execution cost of tasks. The average execution cost minimization by IGSO was $16\% - 43\%$ less than that of GSO for 100 through 500 instances of tasks respectively. The average execution cost minimization by IMFGSO was $15\% - 45\%$ less than that of MFGSO for 100 through 500 instances of tasks respectively. Moreover, IGA outperforms all algorithms in terms of minimizing execution cost in both workloads for most number of tasks. In addition, the execution cost of tasks increases over increasing number of tasks.

Makespan

Makespan or the completion time is the time when the execution of last task is finished. Measuring the makespan is important as minimizing makespan will help to minimize the EC and meets the deadline of task.

Figures 4a and Figure 4b show the results of makespan in the random and NASA Ames iPSC/860 workloads for GSO, MFGSO, GA, IGSO, IMFGSO and IGA algorithms.

Assuming the number of VMs is fixed as 4 VMs and the number of tasks is gradually increased from 100 to 500 tasks. The y axis shows the effect on makespan of increasing the number of tasks as shown in Figure 4a and Figure 4b.

It can be seen from Figure 4a and Figure 4b that the makespan increases over increasing number of tasks. In addition, JTSVMP can improve the performance of algorithms in terms of minimizing the makespan. IMFGSO outperforms all other algorithms in minimizing the makespan in both types of workload. The average makespan minimization by IGSO was $4\% - 14\%$ less than that of GSO for 100 through 500 instances of tasks respectively. The average makespan minimization by IMFGSO was $5\% - 17\%$ less than that of MFGSO for 100 through 500 instances of tasks respectively.

Degree of Imbalance (DoI)

The degree of imbalance measures the imbalance among VMs. It describes the amount of load distribution among the VMs regarding to their execution competencies. It can be calculated as in Eq. 16 below.

$$DoI = \frac{CT_{ij}max - CT_{ij}min}{CT_{ij}avg} \tag{16}$$

where $CT_{ij}max$, $CT_{ij}min$ and $CT_{ij}avg$ are the maximum, minimum and average completion time of executing task i among total VMs.

The small value of DoI informs that the load of the system is more balanced and efficient. The average degree of imbalance of each algorithm with the number of tasks varying from 100 to 500 in random and real workloads is shown in Figure 5a and Figure 5b. It can be seen from Figure 5a and Figure 5b that JTSVMP leads to improve the performance in terms of VMs load balancing. IGSO, IMFGSO and IGA outperformed GSO, MFGSO and GA, respectively in terms of minimizing the DoI.

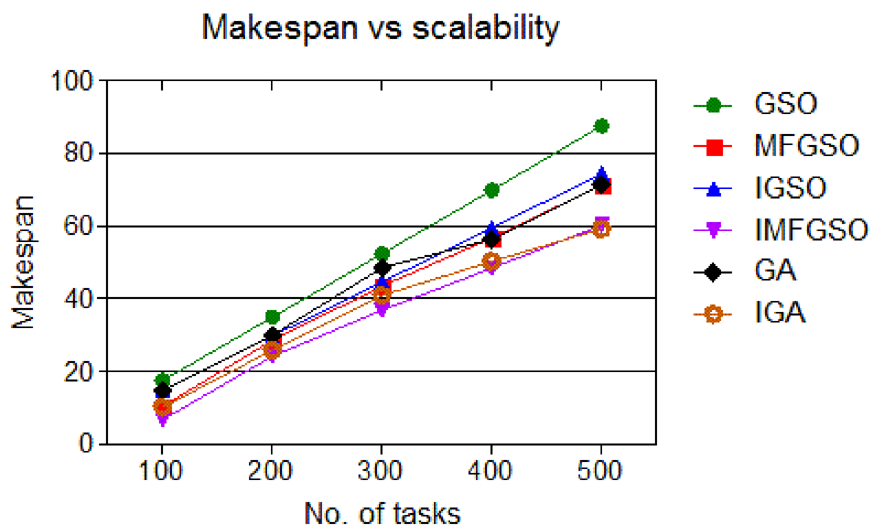
Resource Utilization

This metric shows how PHs is utilized as maximizing the PHs utilization is preferred. Figures 6a and 6b show the results of the resource utilization of PHs in the random and NASA Ames iPSC/860 workloads for GSO, MFGSO, GA, IGSO, IMFGSO and IGA algorithms.

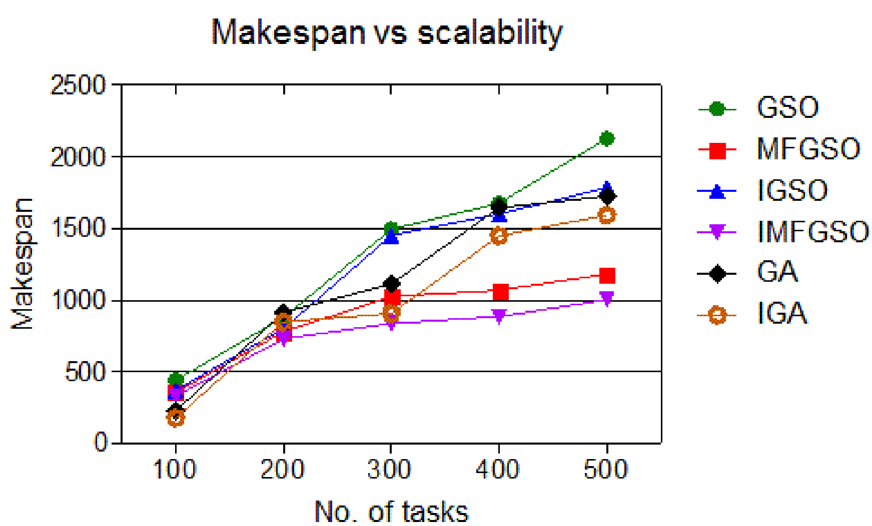
Assuming the number of VMs is fixed as 4 VMs and the number of tasks is fixed to 500 tasks. The y axis shows the effect on resource utilization of PHs of different algorithms as shown in Figures 6a and 6b. It can be seen from Figure 6a and Figure 6b that IGSO, IMFGSO and IGA have better resource utilization of PHs than the GSO, MFGSO and GA, respectively, due to the IGSO, IMFGSO and IGA take the resource utilization into consideration when scheduling tasks to the suitable VMs, while GSO, MFGSO and GA consider only the execution cost within the deadline of tasks. Hence, the JTSVMP leads to improve the performance in terms of maximizing the resource utilization of PHs.

Wilcoxon Signed-rank Test

To further evaluate the performance of the metaheuristic algorithms, the non-parametric statistical test Wilcoxons rank-sum test [58] is carried out at 5% significance level to check whether the improvement achieved by the JTSVMP architecture with integrated algorithms (IGSO, IMFGSO and IGA) is statistically significant or not. Wilcoxon signed-rank test is a non-parametric statistical test of non-independent data from only two groups. This test is carried out to examine the null hypothesis that two samples come from the same population (difference in means is equal to 0) against the alternative hypothesis, especially that a population tends to have larger values than the other. The advantage of the Wilcoxon rank sum test, compared to other tests like the t-test, is that it is more robust to outliers and heavy tail distributions.

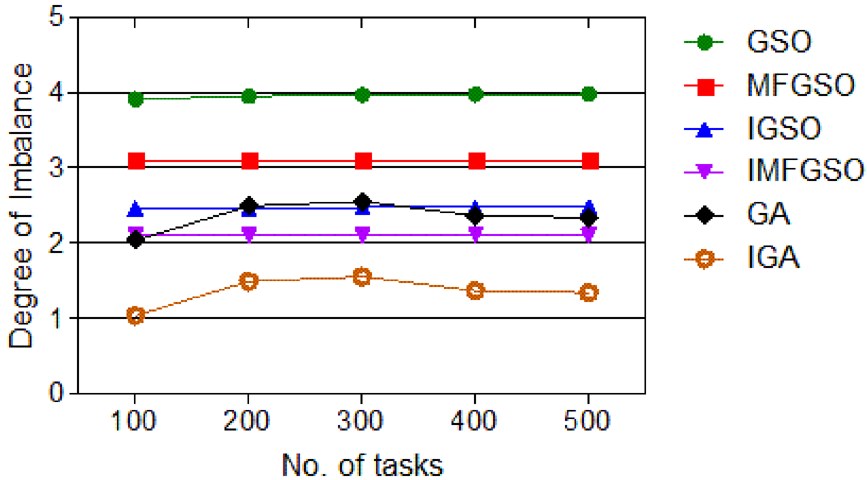


(a) Random workload

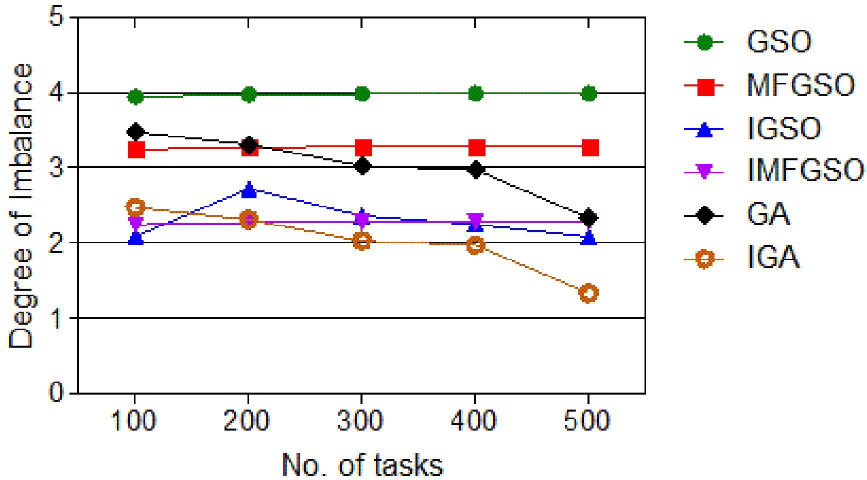


(b) NASA Ames iPSC/860 workload

Figure 4: Makespan

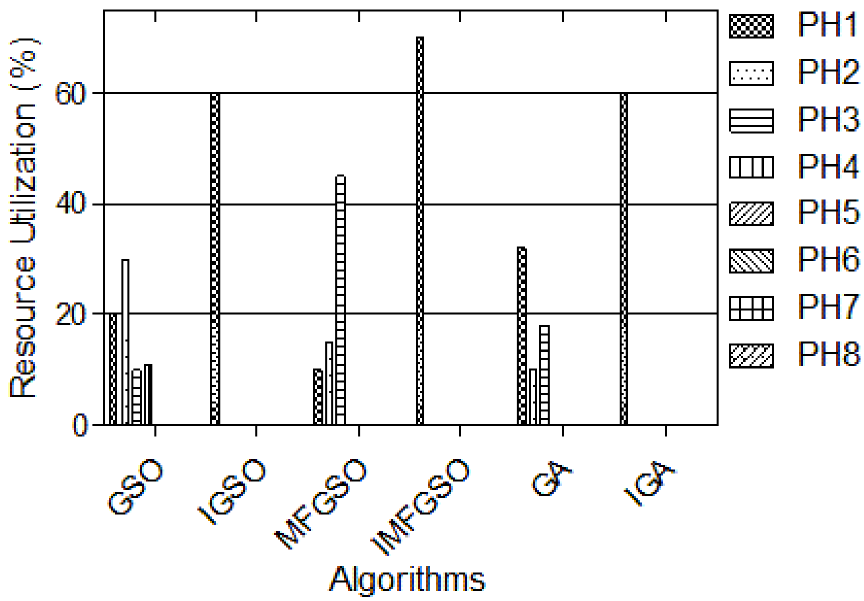


(a) Random workload

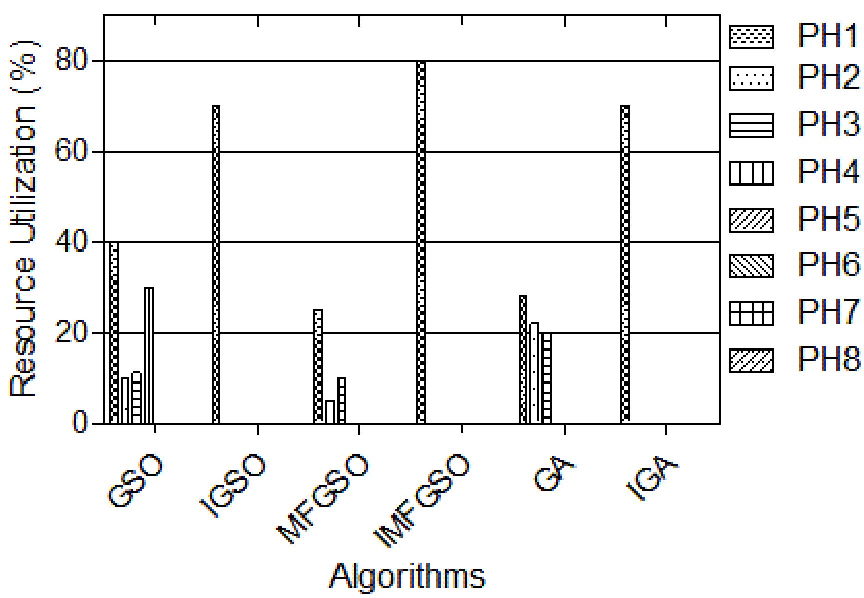


(b) NASA Ames iPSC/860 workload

Figure 5: Degree of Imbalance



(a) Random workload



(b) NASA Ames iPSC/860 workload

Figure 6: Resource utilization

Table 8: P-values of the Wilcoxon test of algorithms based on average execution cost (framed where p $\geq$ 0.05)

Workload	No. of tasks	IGSO vs GSO	IMFGSO vs MFGSO	IGA vs GA
Random	100	$\boxed{0.0555}$	$\boxed{0.0511}$	0.0300
	200	0.0300	0.0300	$\boxed{0.0555}$
	300	0.0200	0.0200	0.0200
	400	0.0200	0.0200	0.0200
	500	0.0200	0.0200	0.0200
NASA Ames iPSC/860	100	$\boxed{0.0611}$	$\boxed{0.0634}$	0.0423
	200	0.0423	0.0320	0.0300
	300	0.0030	0.0300	0.0020
	400	0.0030	0.0300	0.0300
	500	0.0020	$\boxed{0.0510}$	0.0300

Table 8 lists the p values obtained by the test using GraphPad Prism software between algorithms with JTSVMP architecture and with task scheduling only in each workload. Generally, p-values less than 0.05 give a strong evidence against the null hypothesis, which proves the significant difference between algorithms at a level of 5%. It can be observed from Table 8 that the p-values confirm that the improvement made by IGSO over GSO, IMFGSO over MFGSO and IGA over GA are statistically significant for most cases in both workloads in terms of average execution cost.

6. Conclusion and Future Work

In this article, we studied the integration of task scheduling and VM placement problems. MOA was implemented to schedule independent tasks to VMs and place VMs on PHs. Execution cost, makespan, DoI and resource utilization were measured and the integration of task scheduling and VM placement was found to be better than the considering task scheduling only. The JTSVMP integrated algorithms (IGSO, IMFGSO and IGA) have less execution cost and less makespans than the GSO, MFGSO and GA. The average execution cost minimization by IGSO was $16\% - 43\%$ less than that of GSO for 100 through 500 instances of tasks respectively. The average execution cost minimization by IMFGSO was $15\% - 45\%$ less than that of MFGSO for 100 through 500 instances of tasks respectively. The average makespan minimization by IGSO was $4\% - 14\%$ less than that of GSO for 100 through 500 instances of tasks respectively. The average makespan minimization by IMFGSO was $5\% - 17\%$ less than that of MFGSO for 100 through 500 instances of tasks respectively. The JTSVMP integrated algorithms (IGSO, IMFGSO and IGA) also have higher resource utilization of PHs than the GSO and MFGSO. Possible future research may investigate more aspects of the JTSVMP such as security and reliability.

References

[1] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality

for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599–616, 2009.

[2] Cisco. Cisco global cloud index: Forecast and methodology, 2016–2021 white paper, Nov. 2018.

[3] Mohit Kumar, SC Sharma, Anubhav Goel, and SP Singh. A comprehensive survey for scheduling techniques in cloud computing. *Journal of Network and Computer Applications*, 2019.

[4] Mohammad Masdari, Sayyid Shahab Nabavi, and Vafa Ahmadi. An overview of virtual machine placement schemes in cloud computing. *Journal of Network and Computer Applications*, 66:106–127, 2016.

[5] Norman Bobroff, Andrzej Kochut, and Kirk Beaty. Dynamic placement of virtual machines for managing sla violations. In *Integrated Network Management, 10th IFIP/IEEE Int. Symp. on*, pages 119–128. IEEE, 2007.

[6] Thiago AL Genez, Luiz F Bittencourt, and Edmundo RM Madeira. Workflow scheduling for saas/paas cloud providers considering two sla levels. In *IEEE Network Operations and Management Symp.*, pages 906–912. IEEE, 2012.

[7] Brendan Jennings and Rolf Stadler. Resource management in clouds: Survey and research challenges. *Journal of Network and Sys. Management*, 23(3):567–619, 2015.

[8] Marco Polverini, Antonio Cianfrani, Shaolei Ren, and Athanasios V Vasilakos. Thermal-aware scheduling of batch jobs in geographically distributed data centers. *IEEE Transactions on cloud computing*, 2(1):71–84, 2013.

[9] Guiyi Wei, Athanasios V Vasilakos, Yao Zheng, and Naixue Xiong. A game-theoretic method of fair resource allocation for cloud computing services. *The journal of supercomputing*, 54(2):252–269, 2010.

[10] Fei Xu, Fangming Liu, Hai Jin, and Athanasios V Vasilakos. Managing performance overhead of virtual machines in cloud computing: A survey, state of the art, and future directions. *Proceedings of the IEEE*, 102(1):11–31, 2013.

[11] Jixian Zhang, Xutao Yang, Ning Xie, Xuejie Zhang, Athanasios V Vasilakos, and Weidong Li. An online auction mechanism for time-varying multidimensional resource allocation in clouds. *Future Generation Computer Systems*, 2020.

[12] Lena Mashayekhy, Mahyar Movahed Nejad, Daniel Grosu, and Athanasios V Vasilakos. An online mechanism for resource allocation and pricing in clouds. *IEEE transactions on computers*, 65(4):1172–1184, 2015.

[13] Chenhong Zhao, Shanshan Zhang, Qingfeng Liu, Jian Xie, and Jicheng Hu. Independent tasks scheduling based on genetic algorithm in cloud computing. In *5th Int. Conf. on Wireless Communications, Networking and Mobile Comp.*, pages 1–4. IEEE, 2009.

[14] Sung Ho Jang, Tae Young Kim, Jae Kwon Kim, and Jong Sik Lee. The study of genetic algorithm-based task scheduling for cloud computing. *Int. Journal of Control and Automation*, 5(4):157–162, 2012.

[15] Miao Zhang, Yang Yang, Zhenqiang Mi, and Zenggang Xiong. An improved genetic-based approach to task scheduling in inter-cloud environment. In *Ubiquitous Intelligence and Comp. and IEEE 12th Intl. Conf. on Autonomic and Trusted Comp. and IEEE 15th Intl. Conf. on Scalable Comp. and Communications and Its Associated Workshops, IEEE 12th Intl. Conf. on*, pages 997–1003. IEEE, 2015.

[16] Kai Zhu, Huaguang Song, Lijing Liu, Jinzhu Gao, and Guojian Cheng. Hybrid genetic algorithm for cloud computing applications. In *Services Comp. Conf., IEEE Asia-Pacific*, pages 182–187. IEEE, 2011.

[17] Kun Li, Gaochao Xu, Guangyu Zhao, Yushuang Dong, and Dan Wang. Cloud task scheduling based on load balancing ant colony optimization. In *6th Annual ChinaGrid Conf.*, pages 3–9. IEEE, 2011.

[18] Medhat A Tawfeek, Ashraf El-Sisi, Arabi E Keshk, and Fawzy A Torkey. Cloud task scheduling based on ant colony optimization. In *Comp. Engineering & Sys., 8th Int. Conf. on*, pages 64–69. IEEE, 2013.

[19] Weifeng Sun, Ning Zhang, Haotian Wang, Wenjuan Yin, and Tie Qiu. Paco: A period aco based scheduling algorithm in cloud computing. In *Cloud Comp. and Big Data (CloudCom-Asia), Int. Conf. on*, pages 482–486. IEEE, 2013.

[20] Zhanghui Liu and Xiaoli Wang. A pso-based algorithm for load balancing in virtual machines of cloud computing environment. In *Int. Conf. in Swarm Intelligence*, pages 142–147. Springer, 2012.

[21] Shaobin Zhan and Hongying Huo. Improved pso-based task scheduling algorithm in cloud computing. *Journal of Info. & Computational Science*, 9(13):3821–3829, 2012.

[22] Hussein S Al-Olimat, Mansoor Alam, Robert Green, and Jong Kwan Lee. Cloudlet scheduling with particle swarm optimization. In *Communication Sys. and Network Tech., 5th Int. Conf. on*, pages 991–995. IEEE, 2015.

[23] Brototi Mondal, Kousik Dasgupta, and Paramartha Dutta. Load balancing in cloud computing using stochastic hill climbing-a soft computing approach. *Procedia Tech.*, 4:783–789, 2012.

[24] Gamal F Elhady and Medhat A Tawfeek. A comparative study into swarm intelligence algorithms for dynamic tasks scheduling in cloud computing. In *IEEE 7th Int. Conf. on Intelligent Comp. and Info. Sys.*, pages 362–369. IEEE, 2015.

[25] Yangyang Dai, Yuansheng Lou, and Xin Lu. A task scheduling algorithm based on genetic algorithm and ant colony optimization algorithm with multi-qos constraints in cloud computing. In *Intelligent Human-Machine Sys. and Cybernetics, 7th Int. Conf. on*, volume 2, pages 428–431. IEEE, 2015.

[26] Mohammed Abdullahi, Md Asri Ngadi, et al. Symbiotic organism search optimization based task scheduling in cloud computing environment. *Future Generation Computer Sys.*, 56:640–650, 2016.

[27] Mohamed Amine Kaaouache and Sadok Bouamama. Solving bin Packing Problem with a Hybrid Genetic Algorithm for VM Placement in Cloud. *Procedia Computer Science*, 60:1061–1069, 2015.

[28] Jing Xu and Jose AB Fortes. Multi-objective virtual machine placement in virtualized data center environments. In *Green Comp. and Communications, IEEE/ACM Int. Conf. on & Int. Conf. on Cyber, Physical and Social Comp.*, pages 179–188. IEEE, 2010.

[29] Chao Liu, Chenyang Shen, Sitian Li, and Sinong Wang. A new evolutionary multi-objective algorithm to virtual machine placement in virtualized data center. In *Software Eng. and Service Science, 5th IEEE Int. Conf. on*, pages 272–275. IEEE, 2014.

[30] Shahram Jamali and Sepideh Malektaji. Improving grouping genetic algorithm for virtual machine placement in cloud data centers. *4th Intl. Conf. on Computer and Knowledge Eng.*, pages 328–333, 2014.

[31] Christina Terese Joseph, K Chandrasekaran, and Robin Cyriac. Improving the efficiency of genetic algorithm approach to virtual machine allocation. In *Computer and Communication Tech., Int. Conf. on*, pages 111–116. IEEE, 2014.

[32] Yongqiang Wu, Maolin Tang, and Warren Fraser. A simulated annealing algorithm for energy efficient virtual machine placement. *IEEE Intl. Conf. on Sys., Man, and Cybernetics*, pages 1245–1250, 2012.

[33] Nima Khalilzad, Hamid Reza Faragardi, and Thomas Nolte. Towards energy-aware placement of real-time virtual machines in a cloud data center. In *High Performance Comp. and Communications, IEEE 7th Int. Symp. on CSS, IEEE 12th Int. Conf. on ICESS, IEEE 17th Int. Conf. on*, pages 1657–1662. IEEE, 2015.

[34] H. M. Ali and Daniel C. Lee. A biogeography-based optimization algorithm for energy efficient virtual machine placement. *IEEE Symp. on Swarm Intelligence*, pages 1–6, 2014.

[35] Qinghua Zheng, Rui Li, Xiuqi Li, and Jie Wu. A Multi-objective Biogeography-Based Optimization for Virtual Machine Placement. *15th IEEE/ACM Intl. Symp. on Cluster, Cloud and Grid Comp.*, pages 687–696, 2015.

[36] Shangguang Wang, Zhipiao Liu, Zibin Zheng, Qibo Sun, and Fangchun Yang. Particle swarm optimization for energy-aware virtual machine placement optimization in virtualized data centers. *Intl. Conf. on Parallel and Dist. Sys.*, pages 102–109, 2013.

[37] An-ping Xiong and Chun-xiang Xu. Energy Efficient Multiresource Allocation of Virtual Machine Based on PSO in Cloud Data Center. *Mathematical Problems in Eng.*, 2014:1–8, 2014.

[38] Jipeng Gao and Gaoming Tang. Virtual Machine Placement Strategy Research. *Intl. Conf. on Cyber-Enabled Dist. Comp. and Knowledge Discovery*, (2):294–297, 2013.

[39] Seyed Ebrahim Dashti and Amir Masoud Rahmani. Dynamic VMs placement for energy efficiency by PSO in cloud computing. *Journal of Experimental & Theoretical Artificial Intelligence*, pages 1–16, 2015.

[40] E Feller, L Rilling, and C Morin. Energy-Aware Ant Colony Based Workload Placement in Clouds. *Grid Comp., 12th IEEE/ACM Intl. Conf. on*, pages 26–33, 2011.

[41] Xiao-Fang Liu, Zhi-Hui Zhan, Ke-Jing Du, and Wei-Neng Chen. Energy aware virtual machine placement scheduling in cloud computing based on ant colony optimization approach. In *Proc. of the conf. on Genetic and evolutionary computation*, pages 41–48. ACM, 2014.

[42] Fahimeh Farahnakian, Adnan Ashraf, and Tapio Pahikkala. Using Ant Colony System to Consolidate VMs for Green Cloud Computing. *IEEE Trans. on Services Comp.*, 8(2):187–198, 2015.

[43] Md Hasanul Ferdaus, Manzur Murshed, Rodrigo N Calheiros, and Rajkumar Buyya. Virtual machine consolidation in cloud data centers using aco metaheuristic. In *Euro-Par Parallel Processing*, pages 306–317. Springer, 2014.

[44] Yongqiang Gao, Haibing Guan, Zhengwei Qi, Yang Hou, and Liang Liu. A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. *Journal of Computer and Sys. Sciences*, 79(8):1230–1242, 2013.

[45] Medhat A Tawfeek, Ashraf B El-Sisi, Arabi E Keshk, and Fawzy A Torkey. Virtual machine placement based on ant colony optimization for minimizing resource wastage. *Communications in Computer and Info. Science*, 488:153–164, 2014.

[46] Mohammadhossein Malekloo and Nadjia Kara. Multi-objective ACO virtual machine placement in cloud computing environments. *IEEE Globecom Workshops*, pages 112–116, 2014.

[47] Jian Kang Dong, Hong Bo Wang, Yang Yang Li, and Shi Duan Cheng. Virtual machine placement optimizing to improve network performance in cloud data centers. *Journal of China Universities of Posts and Tele.*, 21(3):62–70, 2014.

[48] Antonio Marotta and Stefano Avallone. A simulated annealing based approach for power efficient virtual machines consolidation. In *Cloud Comp., IEEE 8th Int. Conf. on*, pages 445–452. IEEE, 2015.

[49] Dabiah Ahmed Alboaneen, Huaglory Tianfield, and Yan Zhang. Moth-flame glowworm swarm optimisation. *Multiagent and Grid Systems*, 15(3):305–326, 2019.

[50] Dabiah Ahmed Alboaneen, Huaglory Tianfield, and Yan Zhang. Glowworm swarm optimisation algorithm for virtual machine placement in cloud computing. In *Ubiquitous Intelligence & Comp., Advanced and Trusted Comp., Scalable Comp. and Communications, Cloud and Big Data Comp., Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), Intl IEEE Conf.*, pages 808–814. IEEE, 2016.

[51] Dabiah Alboaneen, Huaglory Tianfield, and Yan Zhang. Glowworm swarm optimisation based task scheduling for cloud computing. In *Proc. of the Int. Conf. on Internet of Things and Cloud Comp.,(Cambridge, 22-23 Mar)*, pages 1–7. ACM, 2017.

[52] Anqi Xu, Yang Yang, Zhenqiang Mi, and Zenggang Xiong. Task scheduling algorithm based on pso in cloud environment. In *Ubiquitous Intelligence and Comp. and IEEE 12th Intl Conf on Autonomic and Trusted Comp. and IEEE 15th Intl Conf on Scalable Comp. and Communications and Its Associated Workshops, IEEE 12th Intl Conf on*, pages 1055–1061. IEEE, 2015.

[53] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.

[54] Dror G Feitelson, Dan Tsafrir, and David Krakov. Experience with using the parallel workloads archive. *Journal of Parallel and Dist. Comp.*, 74(10):2967–2982, 2014.

[55] Mohammed Abdullahi, Md Asri Ngadi, Salihu Idi Dishing, Barroon Isma'eel Ahmad, et al. An efficient symbiotic organisms search algorithm with chaotic optimization strategy for multi-objective task scheduling problems in cloud computing environment. *Journal of Network and Computer Applications*, 133:60–74, 2019.

[56] KN Krishnanand and Debasish Ghose. Glowworm swarm based optimization algorithm for multimodal functions with collective robotics applications. *Multiagent and Grid Sys.*, 2(3):209–222, 2006.

[57] John H Holland. Genetic algorithms and the optimal allocation of trials. *SIAM Journal on Comp.*, 2(2):88–105, 1973.

[58] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83, 1945.