Received: September 23, 2018

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Hybrid Model for Load Balancing and Server Consolidation in Cloud

Geetha Megharaj[1]* **Mohan Kabadi[2]**

[1]Department of Computer Science and Engineering,
Sri Krishna Institute of Technology, Bangalore, Karnataka, India
[2]Department of Computer Science and Engineering,
Presidency University, Bangalore, Karnataka, India
* Corresponding author's Email: geethagvit@yahoo.com

Abstract: One of the most pressing and critical issues in cloud is Power Optimization. Due to the popularity of cloud, many computing applications are being hosted in the cloud. Naturally, many Cloud Centers (CCs) are experiencing huge power consumption problem, which leads to higher operational cost and environmental hazards due to carbon emissions. The two important concepts used to achieve power optimization are: load balancing and server consolidation. The first concept aims to achieve fair distribution of computing load on different Physical Machine (PM); whereas, the second concept aims to shutdown PMs, which are having limited computational load. One of the most popular techniques to achieve load balancing and server consolidation is Virtual Machine (VM) migration technique; where, the VMs are packed inside limited PMs; such that, load is fairly balanced among PMs, and limited usage PMs are relieved of their computational load and can be shutdown. However, VM migration requires excessive cost, and results in excessive computed task wastage. Hence, in this work, a new hybrid scheme which does not migrate VMs, and migrates only suitable tasks from overloaded VMs and lightly loaded PMs is presented. The proposed hybrid model is compared against contemporary VM migration technique to assess resource usage merits. The proposed hybrid model significantly outperforms the contemporary VM migration techniques in resource usage efficiency.

Keywords: Cloud computing, Load balancing, Virtual machine, Task migration, Server consolidation.

1. Introduction

The usage and impact of Cloud Computing is exploding contemporarily. The most significant advantage of Cloud Computing is on-demand service availability; wherein, enterprises can demand computing resources--when-needed, where-needed and duration-needed. Due to this advantage, enterprises are relieved from procuring and maintaining expensive computational resources. In-fact, Cloud Computing has provided extensive business opportunities in establishing CCs. However, due to the wide and ever-growing popularity of Cloud Computing, CCs are catering to extremely large computational load, which has resulted in abnormal power utilization leading to excessive costs borne by the CCs.

In the CC nomenclature, the computational devices are called as Physical Machines (PM). Each PM, usually has the capability to cater multiple users. Hence, to provide resource division, and creating the necessary abstraction to the multiple users of each PM, the concept of Virtual Machine (VM) is utilized. Each user is typically allocated a single or group of dedicated VMs. The resource division among different VMs hosted in a single PM might be mutually exclusive. The concept of overloaded PM/VM indicates that, the overloaded PM/VM is currently executing computational tasks which are beyond its capacity to provide efficient execution. In such scenario, all such computational tasks have to share meagerly available resources, which eventually lead to poor task execution efficiency. Similarly, concept of lightly loaded PM indicates that, the corresponding PM is hosting computationally light tasks which exhibit way-below resource usage than the available resources in the PM.

The overloading issue in CCs can result in extensive squandering of computational power; along with, limiting execution efficiency. To address this issue, CCs utilize load sharing mechanism; wherein, computational load is distributed to relieve overloaded entities. Similarly, the issue of lightly loaded PMs can also lead to resource and power squandering in CCs. To address this issue, CCs identify such lightly loaded PMs, and redistribute their computational load to other PMs; so that, such PMs can be shutdown, and this mechanism is denoted as Server Consolidation (SC).

1.1 Research issues

One of the most popular and extensively used load balancing and SC technique in CCs is VM Migration (VMM). Here, all the available VMs in the CC are redistributed into limited number of PMs; such that, both load balancing and SC can be achieved. Myriad of VMM techniques have been proposed in the literature; wherein: approximate algorithms, heuristical and meta-heuristic solutions have been proposed. However, VMM can result in significant performance issues: VMM requires significant memory consumption, and can result in extensive task execution downtime because of stopping the VM for migration; due to VM migration, it is possible that, customer activity information might be lost; VMM might result in significant increase of dirty memory.

In-order to overcome performance issues seen in VMM, VM Task Migration (VMTM) technique was presented in [1]. VMTM involves identifying overloaded VMs, and migrating the extra tasks--which are newly submitted and not yet addressed by the corresponding VM--from the overloaded VMs to other VMs which can host these tasks without getting overloaded. However, VMTM has still not completely addressed power optimization issue in CCs, and many open issues are still prevalent such as: the overloaded VM is not subjected to existing load reduction by identifying and migrating suitable running tasks, and if the running tasks execute for a prolonged period, it can result in significant power squandering; VMTM has still remained elusive w.r.t. SC; VMTM can be extended to address holistic power optimization by combining solutions for: extra task migration, running task migration and SC.

The main goal of this work is to present hybrid model for holistic power optimization using VMTM framework. Compared to the contemporary VMTM technique [1], the proposed hybrid model achieves multiple merits. Firstly, the proposed hybrid model performs runtime task migration along with extra task migration to achieve existing load reduction in overloaded VMs. The identified runtime tasks are selected such that, the tasks are substantially consuming computational resources, and have only completed limited part of their entire execution cycle to ensure that, substantial computational effort is not wasted. Secondly, lightly loaded PMs are identified, and extra tasks submitted to the corresponding VMs are migrated; so that, after the execution completion of all the running tasks, the PMs can be shutdown. Thirdly, the hybrid model, searches for suitable VMTM solutions for both SC and VM overloading issue in a single search procedure. Thereby, reducing the required computational effort when compared to executing separate VMTM procedures to resolve SC and VM overloading issue.

1.2 Contributions

The following contributions are made in this work:

1. Initially, in the hybrid model, the overloaded VMs and lightly loaded PMs are identified through respective discriminant functions. Scoring functions are designed to indicate the value of a specific task migration solution; so that, the most optimal solution can be searched. Separate scoring functions are designed for: overloaded VM extra task migration, overloaded VM existing task migration and migrating extra tasks from VMs belonging to lightly loaded PMs. However, the optimal task migration solution for all the three scoring functions is searched through the aid of single hybrid model, and by using Particle Swarm Optimization (PSO) search solution technique, because the search problem is shown to have non-polynomial complexity. Hence, the proposed PSO based solution executes approximate optimal solution search through meta-heuristic fashion and in polynomial complexity; also, it provides scope for parallelism in-order to accelerate search efficiency.

2. The proposed hybrid model is simulated in MATLAB, and compared against contemporary VMM techniques. The proposed solution outperforms contemporary solutions in multiple metrics such as: power consumption and task execution efficiency.

1.3 Organization of the paper

This paper is organized as follows: Section 2 presents the related work in the addressed area; the hybrid model is outlined in Section 3; Simulation results are presented in Section 4; finally, the work is concluded in Section 5.

2. Related work

VM migration has been one of the popular load balancing techniques in cloud computing. In [2], VM migration technique focused on load balancing in data centers having multi-rooted tree format. In [3], VM migration technique addressed load balancing in distributed cloud centers; wherein, cloud resources are distributed in different geographical location. In [4], rapid migration scheme for VM migration was proposed. As explained above, even though VM migration techniques have demonstrated load balancing efficiency, they suffer from expensive cost of migration and possible task execution latency delays.

Task scheduling for load balancing in distributed systems--including cloud servers--deal with the problem of distributing the submitted task load on available computational units; so that, maximum utilization of these computational units, and substantial reduction in task execution time can be achieved. It must be noted that, task scheduling does not involve evicting already running tasks, and only distributes the newly submitted tasks for efficient computation. Also, overloaded VM problem is usually not addressed in task scheduling, because the task distribution scheme hypothesizes that, overloading will usually not occur.

A novel programming platform for task scheduling in cloud was presented in [5]. Genetic algorithm based task scheduling techniques for cloud was presented in both [6, 7]. Task scheduling technique for geographically distributed cloud centers was presented in [8]. Survey on different load balancing techniques for cloud was presented in [9]. Similarly, survey on meta-heuristic techniques for load balancing in cloud was presented in [10]. In [11], future problems for task scheduling in cloud were comprehensively presented. Dynamic Collaboration in cloud involves collaborative framework through different participating cloud service providers, and in [12], task scheduling in this new framework was presented. In [13] task scheduling technique for IaaS based cloud centers was presented.

Task scheduling technique through user requirement modeling for computational grids—which can also be relevant to cloud--was presented in [14]. Similarly, PSO based task scheduling technique for computational grids and cloud was presented in [15]. Security based task scheduling technique for cloud using Swarm scheduling approach was presented in [16]. Multi objective task scheduling involves achieving multiple goals such as: minimizing task latency, reducing power consumption etc., and this problem for cloud was addressed in [17]. In [18], another multi objective task scheduling technique for cloud using genetic algorithm was presented. In [19], Honey Bee optimization technique for task scheduling in cloud was presented. In [20], task scheduling in computational grids--which can also be extended to cloud--was also achieved through Honey Bee optimization technique. Task scheduling technique for cloud using Ant Colony optimization framework was presented in [21]. In [22], task scheduling for cloud using probabilistic modeling was presented. In [23], task scheduling technique for cloud using specialized bio-inspired algorithm called: Symbiotic Organism Search, was presented. Multi objective task scheduling technique for cloud using Ant Colony optimization framework was presented in [24]. Hybrid task scheduling algorithm for cloud through merging of two techniques namely: Cuckoo search algorithm and Oppositional based learning was presented in [25]. In [26], evolutionary genetic algorithm framework was utilized to achieve task scheduling in cloud. Similarly, fruit fly optimization framework was utilized in [27] to design task scheduling technique in cloud.

Even though, task scheduling is effective in load balancing for cloud, in some scenarios, the estimated resource consumption for a certain task, which is used as critical parameter in task scheduling techniques, can deviate substantially compared to actual resource utilization--which can burgeon rapidly. In such scenarios, VMs can easily become overloaded, and has to be relived from this computational burden. The VM extra task migration techniques presented in [1, 28] achieves load reduction from overloaded VMs through migrating extra tasks. As outlined above, to achieve even better load reduction as achieved in [1, 28], some of the suitable running tasks in the overloaded VMs need to be identified and migrated--along with extra tasks.

Extensive contributions have been made to achieve SC through VM migration technique. Various techniques for SC in virtualized data center has been discussed in [29]. In [30], two VM migration techniques namely--*Hybrid* and *Dynamic Round Robin*(DRR) was presented. Two states were defined in the solution framework called--retiring and non-retiring. If a PM contains limited number of active VMs which are about to finish their task, then, the PM is in retiring state, else, it is in non-retiring state. The retiring PMs will not accept new tasks, and the active VMs are migrated to suitable PMs. Both, Hybrid and DRR exhibit excellent performance w.r.t. reducing power consumption in CCs.

Most of the VM migration techniques for SC are modeled through *Bin Packing Problem* (BPP), which is NP-complete. An approximation scheme based on *First Fit Decreasing* algorithm was proposed [31] to effectively migrate VMs. Each bin is considered as a PM, and the highest priority PMs are subjected to VM migration.

The *Magnet* scheme proposed in [32], performs selection of suitable subsets of available PMs which can guarantee the expected performance levels. The PMs outside the selected subset are shutdown.

A CC management tool was presented in [33]. This tool not only provides continuous monitoring facility, it also provides facility to perform live migration of VMs.

In [34], it was emphasized that, VMs can be broadly classified as data intensive or CPU intensive based on their respective workloads. For this new framework, the BPP was modified, and suitable approximation schemes were presented.

The placement of migrated VMs for SC was performed through assigning priority levels to the candidate PMs in [35]. The PMs which consume low power were given higher priority.

Non-migratory technique for reduction of power consumption in CCs was presented in [36]. Energy efficiency model and corresponding heuristics were proposed to reduce power consumption in CCs. Similar techniques were presented in [37] which utilized green computing framework.

Resource scheduling techniques for SC were presented in [38]. Here, a new architectural model was presented to calculate energy expenditure for different resource scheduling strategies.

All the described VM migration techniques, even though they achieve noticeable performance in reducing power consumption, they all suffer from excessive down times in completing VM migration, and increase in dirty memory as explained before.

3. Hybrid model for load balancing and SC

3.1 Data Overloaded VM identification scheme

Let, VM_y indicate the y^{th} VM, c_y indicates the number of computing node in VM_y, m_y indicates the memory capacity of VM_y, t_{iy} indicates the i^{th} task present inside VM_y, c_{iy} is the CPU utilization ratio of t_{iy}, if t_{iy} is running on multiple CPUs, then, c_{iy} is the sum of CPU utilization ratio for every CPU on which t_{iy} is being executed, m_{iy} represents the memory utilization ratio of t_{iy} and p_{iy} represents the power consumption of t_{iy}, which is represented in Eq. (1)

$$p_{iy} = c_{iy} \times m_{iy} \tag{1}$$

The total power consumed by all the tasks present in VM_y is indicated by the variable p_y and it is represented in Eq. (2) Here, n_y represents the total number of tasks that are being executed in VM_y.

$$\boldsymbol{p}_y = \frac{\sum_{j=1}^{ny} p_{jy}}{c_y} \tag{2}$$

Two thresholds are defined to detect overloaded VMs. The first threshold is defined over CPU utilization ratio, which is indicated by T_c. The second threshold is defined over power consumption, which is indicated by T_p. The VM_y is decided as overloaded if the value of the function *overloaded* $(VM_y)=1$, otherwise if, *overloaded*$(VM_y) = 0$, then, VM_y is decided as not-overloaded. This case is represented in Eq. (3)

$$Overloaded(VMy) = \begin{cases} 1, \ if\ T_c \le \frac{\sum_{j=1}^{ny} c_{jy}}{c_y} \\ \quad or \\ \quad T_p \le py \\ \\ 0, \quad otherwise \end{cases} \tag{3}$$

3.2 Extra task migration framework

In this framework, the extra tasks of the identified overloaded VMs are subjected to migration to other VMs, in-order to relieve the overloaded VMs from processing these tasks. Consider the scenario where the ith extra task of VM_y indicated by t_{iy} is considered for migrating to VM_z. The benefit of this migration scenario is modeled through a score function represented in Eq. (4). Here, $score_{ET}(t_{iy},VM_z)$ indicates the benefit score of the considered migration; lower the score, better will be the migration scenario; $exeET_{iz}$ indicates the predicted execution time of t_{iy}-- when t_{iy} is migrated to VM_z, and this metric is represented in Eq. (5); $transfer(t_{iy},VM_z)$ indicates the cost of transferring t_{iy} to VM_z, and this metric is represented in Eq. (6); the size of execution data used by t_{iy} is indicated by d_{iy};

the available bandwidth for transferring data between VM_y and VM_z is indicated by bw_{yz}.

$$scoreET(t_{iy}, VM_z) = exeET_{iz} + transfer(t_{iy}, VMz) \quad (4)$$

$$exeET_{iz} = \frac{d_{iy}}{c_z + m_z} \quad (5)$$

$$transfer(t_{iy}, VMz) = \frac{d_{iy}}{bw_{yz}} \quad (6)$$

After identifying the overloaded VMs, the set of extra tasks from these VMs indicated by $[t_{i1y1}, t_{i2y2}, \ldots t_{isys}]$ has to be migrated to suitable VM set. Consider a candidate solution indicated by S_1, having the VM set indicated by $[VM_{z1}, VM_{z2}, \ldots VM_{zs}]$, which can provide feasible migration to these extra tasks. Here, $t_{ij}y_j((1 \le j \le s))$ is considered for migrating to VM_{zj}, and there is no restriction that, the VMs in the set $[VM_{z1}, VM_{z2}, \ldots VM_{zs}]$, have to be distinct. The benefit of this migration scenario is modeled through scoring function represented in Eq. (7). Here, $migration\_score_{ET}(S_1)$ represents the migration score.

$$migration\_scoreET(S1) = \frac{\sum_{j=1}^{s} scoreET(t_{ij}, VM_{zj})}{s} \quad (7)$$

Theorem 1**.** *Minimizing the migration score function represented in Eq. (7) provides the optimal candidate solution for overloaded VMs extra task migration problem.*

Proof. Suppose S_o is the candidate solution obtained by minimizing the migration scoring function, and S_r is the optimal candidate solution. Let's assume: $S_o \neq S_r$ and $S_o > S_r$. Since, the migration scoring function belongs to the class of monotonically non-decreasing functions, S_o cannot be lesser than S_r due to the property of such class of functions

Theorem 2. *The searching problem to find the best candidate solution for the migration scenario represented in Eq. (7) has non-polynomial time complexity.*

Proof. Consider the number of candidate solutions possible for a particular migration scenario; wherein, r $(r<s)$ tasks have to be migrated to VM_{y1}, and s-r tasks have to be migrated to VM_{y2}. Clearly, the number of feasible candidate solutions is given by $\binom{s}{r}$. Now, the considered migration scenario is one among many possible such scenarios. Hence, the complexity of this search problem is $> \binom{s}{r}$, which immediately proves the Theorem.

Theorem 1 indicates that, performing optimal migration of extra tasks of overloaded VMs can be achieved through the minimization of migration score function represented in Eq. (7). Theorem 2. proves that, searching for the optimal candidate solution to migrate extra tasks of overloaded VMs requires non-polynomial time complexity. Hence, in the scenario where very large number of candidate solutions is available, usage of approximate algorithms which provide near-optimal solutions in polynomial time complexity is justified.

3.3 Running task migration framework

To select the suitable running tasks for migration, it is important to select those tasks which have completed executing only small portion of their data. The task completion ratio of t_{iy} is represented in Eq. (4). Here, *task_completion* (t_{iy}) indicates the task completion ratio of t_{iy}, d_{iy} is the size of data used by t_{iy} and $\hat{d}_{iy}$ indicates the size of data already consumed by t_{iy}.

$$task\_completion(t_{iy}) = \frac{\hat{d}_{iy}}{d_{iy}} \quad (8)$$

The suitable tasks for migration are identified through their task completion ratio, CPU utilization and consumed power. The selected task should have the task completion ratio within the specified threshold indicated by T_o. This case is represented in Eq. (5). Since, stopping already executing tasks and migrating them into different VMs along with their data, reduces the task execution efficiency, so, only a single task which provides the maximum benefit in load reduction is selected for migration.

$$task\_completion(t_{iy}) \le T_o \quad (9)$$

The task which has the maximum combined value of both CPU utilization ratio and consumed power is selected for migration, and this case is represented in Eq. (6)

$$task\ selected\ for\ migration = max_{t_{iy}}(c_{iy} + p_{iy}) \quad (10)$$

Suppose that, t_{iy} has to be migrated from VM_y and VM_z is one of the possible VM to which t_{iy} has to be migrated. The score of the migration task is represented in Eq. (11). The value of the parameters exe_{iz}, $transfer(t_{iy}, VM_z)$, p_z, $g(T_{cz}, t_{iy})$ and $g(T_{pz}, t_{iy})$ are represented in Eqs. (12), (13), (14), and (15) respectively. Here, *score*(t_{iy}, VM_z) indicates the migration score, exe_{iz} indicates the cost of executing tiy in VM_z, *transfer*(t_{iy}, VM_z) indicates the transfer cost of transferring t_{iy} from VM_y to VM_z, bw_{yz} indicates the bandwidth between VM_y and VM_z and $\hat{p}_z$ is the power consumed by VM_z when task t_{iy} is migrated to VM_z. The functions $g(T_c, t_{iy})$ and $g(T_p, t_{iy})$ ensure that, the migration of t_{iy} from VM_y to VM_z does not cause CPU utilization threshold and power consumption threshold violations.

Consider the situation where the set of tasks $[t_{i1y1}, t_{i2y2}, \ldots t_{isys}]$ which need to be migrated. One of the candidate solution indicated by S_2 is the VM set $[VM_{z1}, VM_{z2}, \ldots VM_{zs}]$, such that, t_{i1y1} will be migrated to VM_{z1}, t_{i2y2} will be migrated to VM_{z2} and so on t_{isys} will be migrated to VM_{zs}. There is no restriction that, the VM_s in the candidate solution set should be distinct. The migration score for this candidate solution is indicated by migration_*scoreRT*(S_2) is represented in Eq. (16). Here, $t_{ijyj} \rightarrow VM_{zj}$ ($1 \leq j \leq s$) indicates that the task t_{ijyj} has already been assigned to VM_{zj} and is being executed inside it. The CPU and memory utilization ratio of t_{ijyj} in VM_{zj} is assumed to be same as observed when t_{ijyj} was executing inside VM_{zj}. The operator **|** is interpreted as such that.

$$scoreRT(t_{iy}, VM_z) = exeRT_{iz} + transfer(t_{iy}, VMz) + \hat{p}_z - (g(T_c, t_{iy}) + g(T_p, t_{iy})) \tag{11}$$

$$exeRT_{iz} = \frac{d_y}{c_z \times c_{iy} + m_{z\times} m_{iy}} \tag{12}$$

$$\hat{p_z} = p_z + \frac{p_{iy}}{c_z} \tag{13}$$

$$g(T_c, t_{iy}) = \begin{cases} T_c - \frac{\sum_{j=1}^{nz} c_{jz} + c_{iy}}{c_z}, & if\ T_c - \frac{\sum_{j=1}^{nz} c_{jz} + c_{iy}}{c_z} > 0 \\ -\infty, & if\ T_c - \frac{\sum_{j=1}^{nz} c_{jz} + c_{iy}}{c_z} \leq 0 \end{cases} \tag{14}$$

$$g(T_p, t_{iy}) = \begin{cases} T_p - (p_z + \frac{p_{iy}}{c_z}), & if\ T_p - (p_z + \frac{p_{iy}}{c_z}) > 0 \\ -\infty, & if\ T_p - (p_z + \frac{p_{iy}}{c_z}) \leq 0 \end{cases} \tag{15}$$

$$migration\_score(S2) = scoreRT(t_{i1y1}, VM_{z1} | t_{i2y2} \rightarrow VM_{z2}, t_{i3y3} \rightarrow VM_{z3}, \ldots . t_{isys} \rightarrow VM_{zs}) + scoreRT(t_{i2y2}, VM_{z2} | t_{i1y1} \rightarrow VM_{z1}, t_{i3y3} \rightarrow VM_{z3}, \ldots . t_{isys} \rightarrow VM_{zs}) + \ldots . . scoreRT(t_{isys}, VM_{zs} | t_{i1y1} \rightarrow VM_{z1}, t_{i2y2} \rightarrow VM_{z2}, \ldots . t_{i(s-1)y(s-1)} \rightarrow VM_{z(s-1)}) \tag{16}$$

Theorem 3. Minimizing the migration score function represented in Eq. (16) provides the optimal candidate solution for overloaded VMs running task migration problem.

Proof. The proof is on the same lines as outlined for Theorem 1.

Theorem 4. The searching problem to find the best candidate solution for the migration scenario represented in Eq. (16) has non-polynomial time complexity.

Proof. The proof is on the same lines as outlined for Theorem 2.

Theorems 3. and 4. prove that, searching for the optimal candidate solution to solve overloaded VMs running task problem, requires approximate and polynomial time complexity algorithms.

3.4 Task migration framework for SC

The first step in SC is to identify suitable PMs which can be considered for shutting down. Let, PM_k indicate the k^{th} PM in the CC, *num*(PM_k) indicate the number of active VMs in PM_k. Each PM is defined with a corresponding threshold indicated by *SD*(PM_k), which indicates the required minimum number of VMs running in the PM to prevent it from shutting down. This case is represented in Eq. (17). Here, *shutdown*(PM_k) = *1* indicates that, PM_k should be shutdown, and *shutdown*(PM_k) = *0* indicates that, PM_k should be kept active.

$$shutdown(PMk) = \begin{cases} 1, & if\ \text{num(PMk)} < \text{SD(PMk)} \\ 0, & otherwise \end{cases} \tag{17}$$

Let, $\widehat{SD}$ indicate the set of PMs which are eligible to be shutdown, and $\widehat{VM}$ indicate the set of active VMs hosted inside those PMs $\in \widehat{SD}$. The extra or new tasks which are submitted to $\widehat{VM}$ will be migrated to other suitable PMs. Once, the running tasks $\widehat{VM}$ finish their execution, all the PMs $\widehat{SD}$ can be shutdown.

Let, t_{iy} indicate the ith extra task submitted to $VM_y \in \widehat{VM}$, and suppose it can be migrated to VM_z which is hosted in that PM $\notin \widehat{SD}$. The migration of t_{iy} also requires the migration of data associated with t_{iy}. The merit of this migration is analyzed through a scoring function represented in Eq. (18). Here, scoreSC(t_{iy}, VM_z) indicates the score of migration strategy which migrates t_{iy} from VM_y to *VMz*, $exeSC_{iz}$ indicates the estimated execution time of t_{iy} inside VM_z, which is represented in Eq. (19).

$$scoreSC(t_{iy}, VM_z) = exeSC_{iz} + transfer(t_{iy}, VMz) \quad (18)$$

$$exeSC_{iz} = \frac{d_{iy}}{c_z + m_z} \quad (19)$$

The extra task migration is performed batch-wise, rather than on a single task in-order to reduce computational overheads. All the extra tasks submitted to $\widehat{VM}$ are batched together for migration. Consider the scenario, where the batch of extra tasks $[t_{i1y1}.....t_{i2y2}, \ldots t_{isys}]$ submitted to $\widehat{VM}$ need to be migrated. Suppose, $[VM_{z1}, VM_{z2}, \ldots\ldots VM_{zs}]$ is a candidate solution for the required migration of tasks, wherein, $t_{ij}y_j$ $(1 \leq j \leq s)$ is considered to be migrated from VM_{yj} to VM_{zj}, and this candidate solution is denoted as S_3. Also, there is no restriction that, the VMs in the candidate solution should be distinct. The score of this migration scheme indicated by *migration_scoreSC(S₃)* is represented in Eq. (20).

$$migration\_scoreSC(S_3) = \frac{\sum_{j=1}^{s} scoreSC(t_{ij}, VM_{zj})}{s} \quad (20)$$

Theorem 5. Minimizing the migration score function represented in Eq.(20) provides the optimal candidate solution for SC problem involving VM task migration.
Proof. The proof is on the same lines as outlined for Theorem 1.

Theorem 6. The searching problem to find the best candidate solution for the migration scenario represented in Eq. (20) has non-polynomial time complexity.
Proof. The proof is on the same lines as outlined for Theorem 2.

Theorems 5 and 6 prove that, searching for the optimal candidate solution to solve SC problem through VM task migration, requires approximate and polynomial time complexity algorithms.

3.5 Hybrid model

The main goal of the hybrid model is to achieve holistic power optimization through combining all the three frameworks namely: extra task migration framework, running task migration framework and task migration framework for SC. The problem instance of the hybrid model is to migrate all the tasks selected for each of the three frameworks. Each candidate solution for the hybrid model is represented through the Candidate Solution Vector (CSV) represented in Eq. (21). Here, S represents a specific CSV; S_1, S_2 and S_3 represent a specific candidate solutions for extra task migration framework, running task migration framework and task migration framework for SC respectively.

$$S = [S_1, S_2, S_3]^T \quad (21)$$

The merit of the CSV S is analyzed through the scoring function indicated by *hybrid_migration_score(S)* represented in Eq. (22).

$$hybrid\_migration\_score(S) = migration\_scoreET(S_1) + migration\_scoreRT(S_2) + migration\_scoreSC(S_3) \quad (22)$$

Theorem 7. Minimizing the migration score function represented in Eq.(22) provides the optimal candidate solution for hybrid model problem.
Proof. Since, the migration score function is a linear combination of individual migration score functions of three different frameworks represented in Eq. (7), (16) and (20), and it is also monotonically non-decreasing function, it will reach its minimum value when the individual score functions reach their minimal value. Since, the minimum value of individual migration score functions correspond to optimal candidate solutions for their respective frameworks--according to Theorems 1,2 and 3, the Theorem immediately follows.

Theorem 8. The searching problem to find the best candidate solution for the migration scenario represented in Eq. (22) has non-polynomial time complexity.
Proof. According to Theorems 2,4 and 6, the searching problem corresponding to Eq. (7), (16) and (20) have non-polynomial time complexity. Since, the migration score function represented in Eq. (22) is a linear combination of individual migration score functions of three different frameworks represented in Eq. (7), (16) and (20), the Theorem immediately follows.

The Theorem 7 indicates that, the problem of finding the optimal CSV for the hybrid model problem represented in Eq. (22) corresponds to minimizing the migration score function represented in Eq. (22). The Theorem 8 indicates that, finding the optimal solution to the hybrid model problem requires non-polynomial time complexity. Hence, approximate algorithms running in polynomial time complexity need to be designed.

3.6 Solution search

The PSO technique is utilized for finding optimal/sub-optimal solution to the hybrid model problem represented in Eq. (22). PSO technique is a meta-heuristic technique [1] which provides an approximate solution -- in polynomial time complexity -- to optimization problems, and it is inspired by the social behavior of birds. The search for optimal solution is carried out by group of particles; wherein, each particle has an exclusive zone in the candidate solution space, and union of all particle zones is equal to the candidate solution space. Each point in the candidate solution space represents a candidate solution vector. The particles are continuously moving in their corresponding candidate solution space to identify the optimal solution, and are involved in continuous communication for exchanging their locally discovered best solution, which in-turn decides the corresponding velocity of the particle for navigation. The particles continue their search until acceptable solution is obtained.

The PSO utilizes multiple search particles, which are collectively involved in discovering near optimal candidate solution for optimization problem. Here r search particles are assumed. The current position of the ith particle at iteration t be $\vec{X}_i(t)$. The position for the next iteration is indicated by $\vec{X}_i(t+1)$, which is calculated as represented in Eq. (23) Here, $\vec{V}_i(t+1)$ indicates the velocity of the ith particle for $t + 1$ iteration, and it is calculated as represented in Eq. (24). Here, D_1 and D_2 indicate the degree of particle attraction towards individual and group success respectively, $\vec{x}_{\text{gbest}}$ and $\vec{x}_{\text{pbesti}}$ indicate the global best solution obtained by all the particles until the current iteration respectively, W indicates a control variable, and $r1, r2 \in [0, 1]$ are the random factors.

$$\vec{X}_i(t+1) = \vec{X}_i(t) + \vec{V}_i(t+1) \quad (23)$$

$$\vec{V}_i(t+1) = W\vec{V}_i(t) + D_1 r_1 (\vec{x}_{pbesti} - \vec{X}_i(t)) + D_2 r_2 (\vec{x}_{gbest} - \vec{X}_i(t)) \quad (24)$$

The proposed PSO based VM task migration technique for load balancing is outlined in Algorithm 1. Here, *initialize_PSO(P)* divides the candidate solution space among the r search particles indicated by $P = [p_1, p_2, \dots\dots p_r]$ and assigns each particle to some arbitrary positions in their corresponding candidate solution space. Each particle calculates its candidate solution for the corresponding current position through *compute_score*($\vec{X}_i(t)$), which utilizes Eq. (23) and Eq. (24). The values for $\vec{x}_{\text{gbest}}$ and $\vec{x}_{\text{pbesti}}$ are calculated through *local_best*($score_i$) and *global_best*(P, $\vec{x}_{pbesti}$) respectively. The particles continue to search until the acceptable solution is found, and which is calculated through *acceptable*($\vec{x}_{gbest}$).

Algorithm 1 PSO Algorithm for VM task migration

```
$P = [p_1, p_2, \dots\dots p_r]$
initialize_PSO (P)
flag = 0
t = 0
While flag = = 0 do
    t = t + 1
    For i=1 to r do
        $score_i$= compute_score ($\vec{X}_i(t)$)
        $\vec{x}_{pbesti}$ = local_best($score_i$)
        $\vec{x}_{gbest}$ = global_best(P, $\vec{x}_{pbesti}$)
        If acceptable($\vec{x}_{gbest}$) then
            flag = 1
        end if
    end for
    t = t + 1
end while
```

3.7 Architectural model

The architectural model for implementing the hybrid model is illustrated in Fig. 1. Here, VM *Meta-data* component provides all the VM specific meta-data required for the hybrid model in the entire CC; *Overloaded VM Selection* component is responsible for identifying the overloaded VMs; similarly, *Lightly Loaded PM Selection* component is responsible for identifying the lightly loaded PMs; *VM Extra Task Selection* component is responsible for identifying the extra tasks from overloaded VMs; *VM Running Task Selection* component is responsible for identifying the running tasks from overloaded VMs for migration; *PM Extra Task Selection* is responsible for identifying extra tasks from the VMs running inside lightly loaded PMs; *Hybrid Scoring* component is responsible for implementing the proposed hybrid scoring function by using the selected tasks from its lower components; *Solution Search* component is responsible for searching the optimal/sub-optimal solution for the hybrid model through the utilized

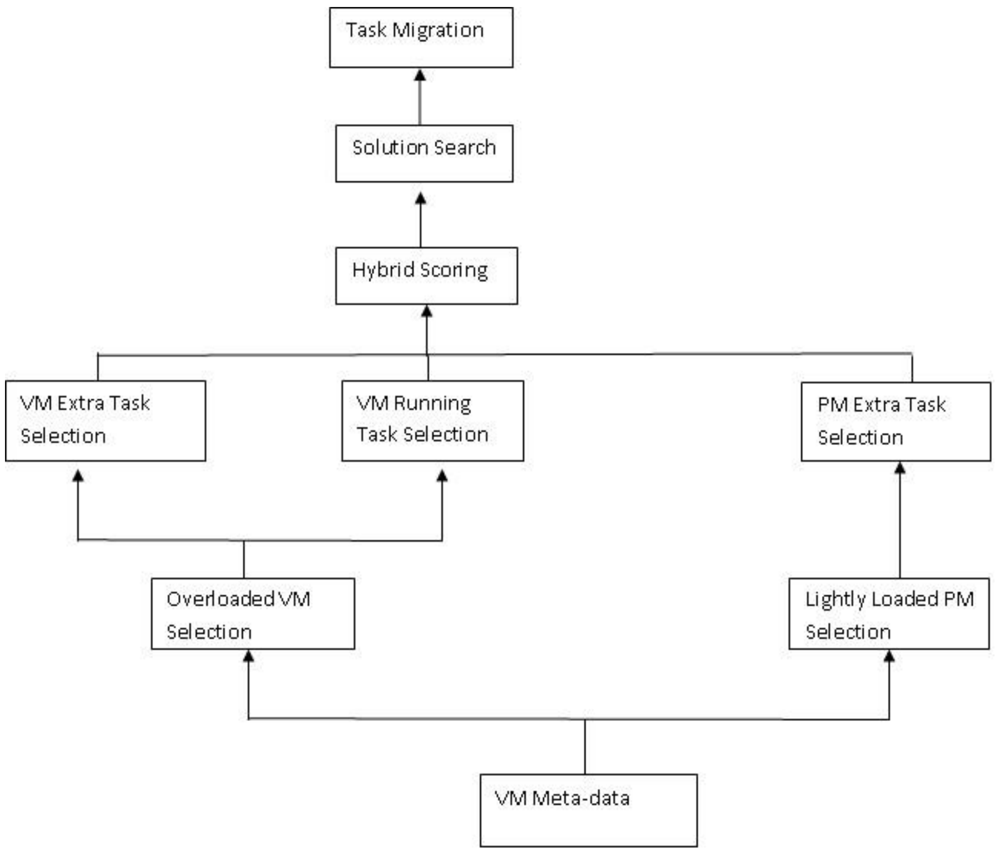

Figure. 1 Architectural model

PSO technique; *Task Migration* component is responsible for migrating the tasks to the intended VMs.

4. Results and discussions

4.1 Simulation setup

The proposed hybrid model is simulated in MATLAB. The simulation parameter settings are presented in Table 1. In-order to exploit parallelism, each PSO particle is assumed to be running on an exclusive computing node. For the ease of reference, the proposed hybrid model is denoted as *HM*.

HM is compared against contemporary VMM techniques presented in [2] and [3], which are -- for the ease of reference -- denoted as VM_M_1 and VM_M_2 respectively. Both VM_M_1 and VM_M_2, utilize the Bin Packing Framework (BPF) for migrating VMs. Here, BPF considers each Bin as a PM with certain resource capacity; each item is considered as a VM. The task is to pack the available items in minimum Bins possible. Naturally, BPF provides load distribution and SC by allotting the available VMs in limited number of PMs. In this simulation study, VMM was initiated as soon as any VM crossed the two thresholds: T_c and T_p. It must be noted that, obtaining optimal solution for BPF is NP-hard. Hence, VM_M_1 and VM_M_2 utilize their customized approximation algorithms.

Totally five performance metrics are defined and utilized for analyzing simulation results. The first metric is denoted as APUR, and which is represented in Eq. (25). Here, APUR indicates the average power utilization ratio inside the CC after execution of either: HM, VM_M_1 or VM_M_2. This metric is calculated by considering all the VMs in the CC.

$$APUR = \frac{\sum_{VM_y \in CC,} P_y}{|CC|} \tag{25}$$

The second metric is indicated as AVEXE, and which is represented in Eq. (26). Here, AVEXE indicates the average task execution time by considering all the running and extra tasks corresponding to every VM in the CC. This metric is calculated after the execution of either: HM, VM_M_1 or VM_M_2.

$$AVEXE = \frac{\sum_{t_i \in CC} exeET_{iz}}{|CC|} \tag{26}$$

Table 1. Simulation parameter settings

Simulation Parameter	**Values**
Number of VMs considered	Varied between 1000- 5000
Number of computing nodes/CPUs in each VM	Varied between 5 to 20
Main memory capacity for each VM	Varied 4GB/ 8GB/16GB
Number of tasks executing in each VM	Varied between 10 to 50 tasks
Bandwidth between any 2 VMs	Varied between 100mbps to 500mbps
CPU utilization ratio for any task	Varied between 0.02 to 0.8
Memory utilization of each task	Varied between 0.02 to 0.8
Number of PSO search particles	Varied between 5 – 25
Number of Computing nodes allotted for each PSO particle	1
Threshold T_c	0.7
Threshold T_p	0.6
Size of task data	Varied between 1GB to 10GB
Threshold T_o	Varied between 0.05 – 0.25
Number of VMs present in each PM indicated by tvm(PM_k)	Varied between 0 to 200 (randomized)
nvm(PM_k)	0.5 × tvm(PM_k)
Number of extra tasks for a VM during Ie	Poisson distributed with λ = 5
min SD(PM_k)	Varied between [5-25]
Power consumed by each VM	Varied between 0 to 1(normalized)

$$AVEXE = \frac{\sum_{t_i \in CC} exeET_{iz}}{|CC|} \tag{26}$$

The third metric is indicated as ARUR, and which is represented in Eq. (27). Here, ARUR indicates the average resource utilization ratio inside the CC after execution of either: HM, VM_M_1 or VM_M_2. This metric is calculated by considering all the VMs in the CC. The metric CPU_y, which is represented in Eq. (28), indicates the average CPU utilization in VM_y , and it is calculated by considering the CPU utilization ratio of every task running inside VM_y .

$$ARUR = \frac{\sum_{VM_y \in CC}, CPU_y}{|CC|} \tag{27}$$

$$CPU_y = \frac{\sum_j c_{jy}}{c_y} \tag{28}$$

The fourth metric is indicated as TCOST, and which is represented in Eq. (29). Here, TCOST represents the total data transfer cost incurred after execution of either: HM, VM_M_1 or VM_M_2. This metric is calculated by considering every task in the CC which was subjected to migration. Here, $U(d_{iy}) = 0$ if t_{iy} is not involved in migration; otherwise, $U(d_{iy}) = 1$.

$$TCOST = \sum_y \sum_i d_{iy} U(d_{iy}) \tag{29}$$

The fifth metric is indicated as ATPT, and which is represented in Eq. (30) and Eq. (31). Here, ATPT represents the average throughput of the CC, in-terms of task executions completed per hour, after execution of either: HM, VM_M_1 or VM_M_2. Every task in the CC is considered for calculating this metric. Here, $\max_z(exeRT_{zy})$ indicates the execution time of t_{zy} , which has the highest execution latency in VM_y . Also, $exeRT_{zy}$ is expressed in-terms of *hours*.

$$ATPT = \frac{\sum_y TPT(VM_y)}{|CC|} \tag{30}$$

$$TPT(VM_y) = \frac{n_y}{\max_z(exeRT_{zy})} \tag{31}$$

4.2 Simulation Results

The first experiment analyzes the performance of *HM, VM_M_1 and VM_M_2* w.r.t. *APUR*. The first experiment has three different cases. In the first case, the number of VMs in the CC is varied. In the second and third case, T_c and T_p are varied respectively. The result of first, second and third case is illustrated in Fig. 2, 3 and Fig. 4 respectively. It is clear that, *HM* outperforms the other two techniques, because VMM techniques try to pack all the VMs in limited or minimal PMs, which leads to higher *APUR*; whereas, *HM* relieves the overloaded VMs and redistributes the tasks to other non-overloaded VMs, ensuring that, there is no decrease in the number of PMs. Hence, *HM* exhibits better *APUR*.

In the first experiment, the performance of *HM* improves with the increase in number of VMs, because with increase in number of VMs, tendency to produce more overloaded VMs also increase. Hence, some of the overloaded VMs might have more running tasks that consume more resources, and their eviction creates more resource release. However, there is little correlation with performance of VMM techniques and number of VMs, because BPF solution is uncorrelated with *APUR* and number of VMs. The performance of *HM* improves with reduction of T_p and T_c, because at lower values of these parameters more number of overloaded VMs come into consideration, which further improves *APUR*, because of more load redistribution. However, lower values of these parameters can trigger frequent load balancing procedures leading to over-all inefficiency in CC functioning. The performance of VMM techniques regarding T_p and T_c again remain uncorrelated for the same reasons explained above.

The second experiment analyzes the performance of *HM*, *VM_M_1* and *VM_M_2* w.r.t. *AVEXE*. The same three cases used in first experiment are also used here. The result of first, second and third case is illustrated in Fig. 5, Fig. 6 and Fig. 7 respectively. The VMM techniques migrate all the existing and extra tasks of every VM. Due to this scenario, many computationally intensive tasks have to be re-executed, which increase *AVEXE*. However, *HM* only migrates a single and resource expensive task, which reduces substantial re-execution latency compared to VMM techniques. Hence, *HM* outperforms other VMM techniques w.r.t. *AVEXE* metric. *HM* exhibits decreasing performance with the increase of number of VMs, because with more number of VMs, more overloaded VMs have to be treated. Thus, more number of tasks get migrated which adds to the existing computational burden of VMs to which tasks have been migrated. Similar reason can be attributed for the performance of VMM techniques.

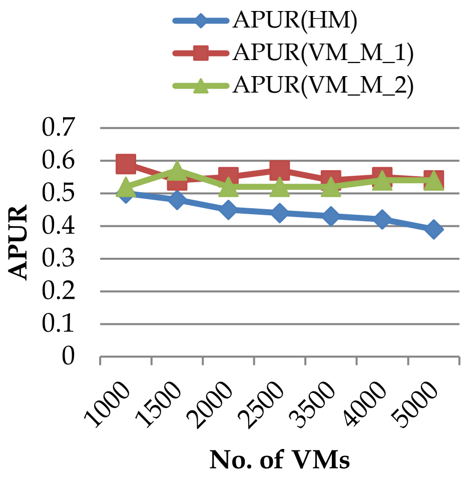

Figure. 2 No. of VMs vs *APUR*

The performance of *HM* worsens with lower values of T_p and T_c, because at lower values of these parameters more number of VMs will be considered as overloaded. Due to this scenario, more number of running tasks will be migrated and re-executed. However, the performance of VMM techniques remain uncorrelated with T_p and T_c for the same reasons explained above.

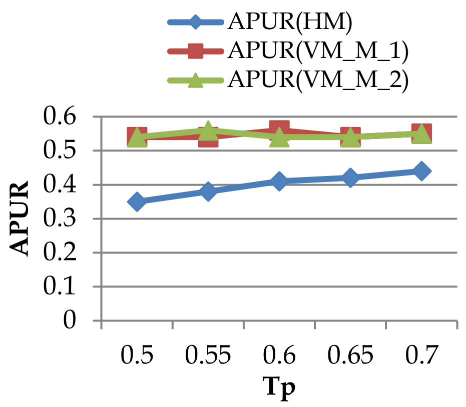

Figure. 3 Tp vs *APUR*

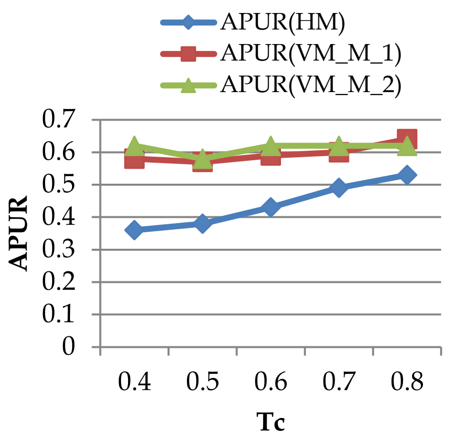

Figure. 4 Tc vs *APUR*

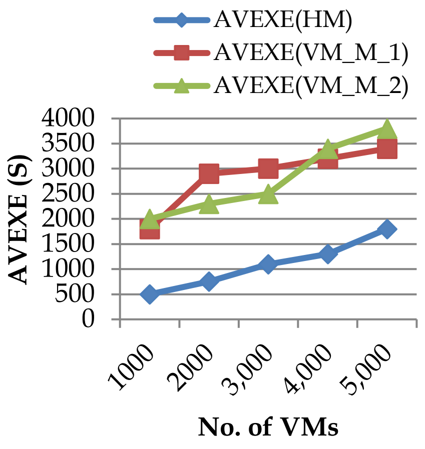

Figure. 5 No of VMs vs AVEXE

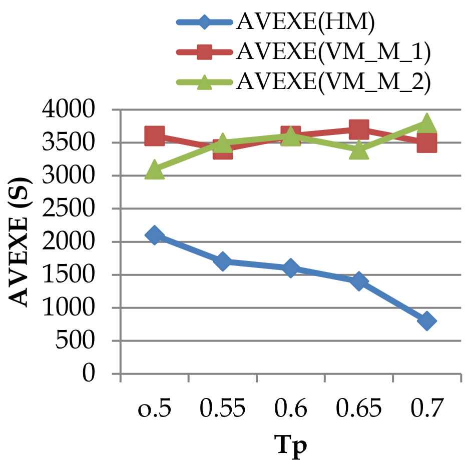

Figure. 6 T_p vs AVEXE

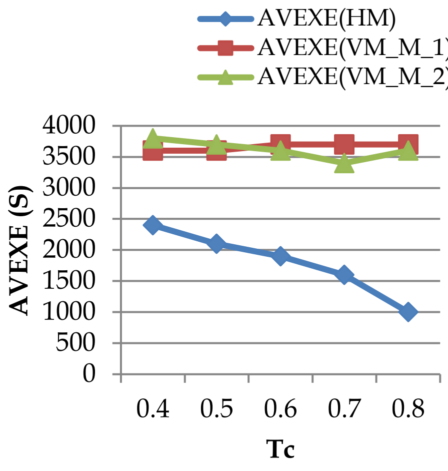

Figure. 7 T_c vs AVEXE

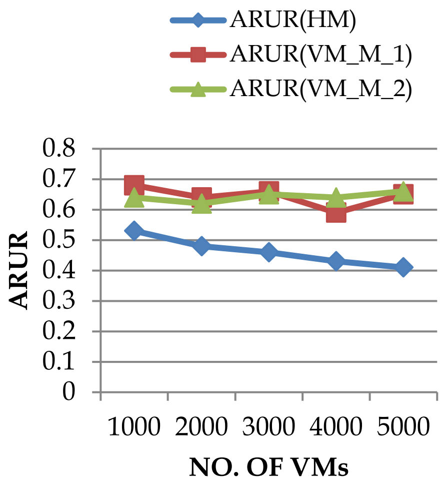

Figure. 8 No. of VMs vs *ARUR*

The third experiment analyzes the relative performance of HM, VM_M_1 and VM_M_2 WRT ARUR, when the number of VMs in the CC is varied. The analysis result of this experiment is illustrated in Fig. 8. Clearly, HM relatively outperforms other techniques. The reasoning for the observed performance of all the three techniques is identical to the reasoning presented for the first case of first experiment.

The fourth experiment analyzes the relative performance of HM, VM_M_1 and VM_M_2 WRT TCOST, when the number of VMs in the CC is varied. The analysis result of this experiment is illustrated in Fig. 9. Again, HM provides the best results. Since, VM_M_1 and VM_M_2 migrate all the VMs in CC, to achieve their goals, all the tasks in CC also get migrated. Hence, TCOST of these VMM techniques is high. The performance of HM decreases slightly as the number of VMs in CC is increased. This performance decrease is due to the fact that, more number of overloaded VMs can get created, when the number of VMs in CC gets increased. Hence, with more number of overloaded VMs, more number of tasks are migrated, which leads to slight increase in TCOST. By using this same reasoning, performance curve exhibited by other two techniques can be described.

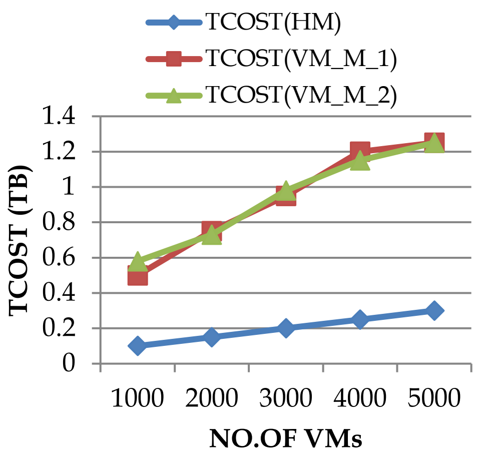

Figure. 9 No. of VMs vs *TCOST*

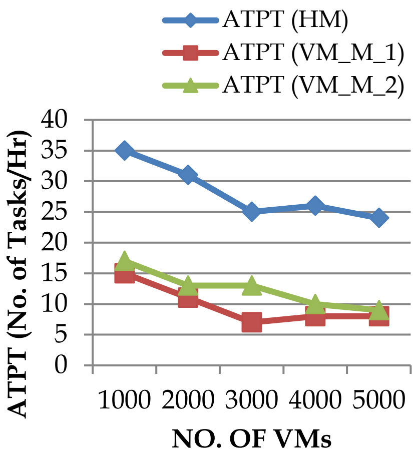

Figure. 10 No. of VMs vs *ATPT*

Table 2. Comparative performance data analysis 1

Experimen t No	Max_PD(HM,VM _M_1) %	Min_PD(HM,VM_ M_1) %
1 (Case 1)	31	8
1 (Case 2)	38	22
1 (Case 3)	38	16
2 (Case 1)	77	70
2 (Case 2)	78	40
2 (Case 3)	73	34
3	35	23
4	76	70
5	64	58

Table 3. Comparative performance data analysis 2

Experiment No	Max_PD(HM,VM _M_2) %	Min_PD(HM,VM _M_2) %
1 (Case 1)	30	3
1 (Case 2)	37	21
1 (Case 3)	40	14
2 (Case 1)	80	72
2 (Case 2)	82	30
2 (Case 3)	70	37
3	37	21
4	79	65
5	66	56

The fifth experiment analyzes the relative performance of HM, VM_M_1 and VM_M_2 WRT ATPT, when the number of VMs in CC is varied. The analysis result of this experiment is illustrated in Fig. 10. Again, HM provides the best performance in-terms of ATPT. As already described in first experiment, the VMM techniques tend to pack more number of tasks in each VM, when compared to HM. Hence, ATPT performance of VMM techniques is poor due to more resource contention. The performance of all the three techniques slightly decreases with the number of VMs, because, as already explained in first case of second experiment, more number of VMs leads to more task migrations, which in-turn leads to higher resource contention.

The comparative performance data analysis study between HM and the considered VMM techniques, is outlined through Tables 2 and 3. Here, *Max_PD(HM,VM_M_1)* and *Min_PD(HM,VM_M_1)* indicate the maximum and minimum percentage performance improvement of HM over VM_M_1 respectively, by considering the metric corresponding to the specific experiment. Similarly, *Max_PD(HM,VM_M_2)* and *Min_PD(HM,VM_M_2)* indicate the maximum and minimum percentage performance improvement of HM over VM_M_2 respectively, by considering the metric corresponding to the specific experiment. From this presented analysis study, it is clear that, HM provides substantial performance improvement over considered VMM techniques, in all the considered performance metrics.

5. Conclusion

In this work, the necessity of utilizing hybrid model for VMTM technique was described; along with, the limitations of VMM and VM extra task migration techniques. The proposed hybrid model utilized three components: extra task migration component, runtime task migration component and SC component; thus, achieving holistic power optimization. The simulation results of the proposed hybrid model were compared against contemporary VMM techniques, and the hybrid model substantially outperformed the contemporary VMM techniques in: power optimization, CPU resource utilization, communication cost, throughput and task execution time. Specifically, the proposed hybrid model provides nearly 30% better benefits according to considered performance metrics against contemporary VMM techniques.

In future, the merits of applying VM task migration schemes for *Distributed Cloud Center* in which, the CC is distributed in different geographical locations need to be analyzed. Task migration in this new setting faces multiple challenges, because migrating tasks to different locations in the cloud center can result in performance limitation due to large geographical distances.

References

[1] F. Ramezani, J. Lu, and F. K. Hussain, “Task Based System Load Balancing in Cloud Computing Using Particle Swarm Optimization”, *International Journal of Parallel Programming*, Vol. 42, No. 5, pp. 739-754, 2014.

[2] N. Jain, I. Menache, J. Naor, and F. Shepherd, “Topology Aware VM Migration in Bandwidth Oversubscribed Datacenter Networks”, In: *Proc. of the 39th International Colloquium*, pp. 586-597, 2012.

[3] S. Kumaraswamy and K. N. Mydhilli, “Virtual Machine Placement in Distributed Cloud Centers using Bin Packing Algorithm”, *International Journal of Grid and Utility Computing*, 2018.

[4] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum, “Optimizing the Migration of Virtual

Computers", In: *Proc. of ACM SIGOPS Oper. Syst. Rev*. 36(SI), 377390, 2002.

[5] A. Whitaker, R. S. Cox, M. Shaw, and S. D. Gribble, "Constructing Services with Interposable Virtual Hardware", In: *Proc. of the 1st Symposium on Networked Systems Design and Implementation*, pp. 169-182, 2004.

[6] Y. A. Zomaya and T. Yee-Hwei, "Observations on Using Genetic Algorithms for Dynamic Load Balancing", *IEEE Transactions on Parallel Distributed Systems*, Vol. 12, No. 9, pp. 899-911, 2001.

[7] C. Zhao, S. Zhang, Q. Liu, J. Xie, and J. Hu, "Independent Tasks Scheduling based on Genetic Algorithm in Cloud Computing", In: *Proc. of the 5th International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 1-4, 2009.

[8] E. Juhnke, T. D. Bock, and D. Freisleben, "Multi Objective Scheduling of BPEL Workflows in Geographically Distributed Clouds", In: *Proc. of the 4th IEEE International Conference on Cloud Computing*, pp. 412-419, 2011.

[9] M. Geetha and K. G. Mohan, "A Survey on Load Balancing Techniques for Cloud Computing", *IOSR Journal of Computer Engineering*, Vol. 18, Issue 2, pp. 55-61, 2017.

[10] S. Poonam, D. Maitreyee, and A. Naveen, "A Review of Task Scheduling Based on Meta-heuristics Approach in Cloud Computing", *Journal Knowledge and Information Systems*, pp. 1-51, 2017.

[11] M. A. Sadeghi and N. N. Jafari, "Load Balancing Mechanisms and Techniques in the Cloud Environments", *Journal of Networks and Computer Applications*, Vol. 71, pp. 86-98, 2016

[12] B. Song, M. M. Hassan, and E. Huh, "A Novel Heuristic-based Task Selection and Allocation Framework in Dynamic Collaborative Cloud Service Platform" In: *Proc. of the 2nd IEEE International Conference on Cloud Computing Technology and Science*, pp. 360-367, 2010.

[13] J. Li, M. Qiu, Z. Ming, G. Quan, X. Qin, and Z. Gu, "Online Optimization for Scheduling Preemptable Tasks on IaaS Cloud Systems", *Journal of parallel and Distributed Computing*, Vol. 72, No. 5, pp. 666-677, 2012.

[14] J. Kolodziej and F. Xhafa, "Modern Approaches to Modeling User Requirements on Resource and Task Allocation in Hierarchical Computational Grids", *International Journal of Applied Mathematics and Computer Science*, Vol. 21, No. 2, pp. 243-257, 2011.

[15] Z. Lei, C. Yuehui, S. Runyuan, J. Shan, and Y. Bo, "A Task Scheduling Algorithm based on PSO for Grid Computing", *Int. J. Comput. Intell. Res.*, No. 4, No. 1, pp. 37-43, 2008.

[16] H. Liu, A. Abrahan, V. Snasel, and S. McLoone, "Swarm Scheduling Approaches for Workflow Applications with Security Constraints in Distributed Data-intensive Computing Environments", *Journal Information Sciences*, pp. 228-243, 2012.

[17] F. Ramezani, J. Lu, and F. Hussain, "Task Based System Load Balancing Approach in Cloud Environments", *Knowledge Engineering and Management*, pp. 31-42, 2014.

[18] F. Ramezani, L. Jie, T. Javid, and H. Farookh Khadeer, "Evolutionary Algorithm-based Multi-objective Task Scheduling Optimization Model in Cloud Environments", *Journal World Wide Web*, Vol. 2015, pp. 1737-1757, 2015.

[19] D. B. LD and P. V. Krishna, "Honey Bee Behavior Inspired Load Balancing of Tasks in Cloud Computing Environments", *Applied Soft Computing Journal*, Vol. 13, No. 5, pp. 2292-2303, 2013.

[20] J. Taheri, Y. L. Choon, A. Y. Zomaya, and H. J. Siegel, "A Bee Colony based Optimization Approach for Simultaneous Job Scheduling and Data Replication in Grid Environments", *Comput. Oper. Res.*, pp. 1564-1578, 2013.

[21] J.-F. Li, J. Peng, X. Cao, and H.-Y. Li, "A Task Scheduling Algorithm based on Improved Ant Colony Optimization in Cloud Computing Environment", *Energy Procedia*, pp. 6833-6840, 2011.

[22] R. Shiva, N. A. Habibizad, R. A. Masoud, and H. Mehdi, "Probabilistic Modeling to Achieve Load Balancing in Expert Clouds", *Ad-Hoc Networks*, pp. 12-23, 2017.

[23] M. Abdullahiac, M. A. Ngadi, and S. M. Abdulhamid, "Symbiotic Organism Search Optimization Based Task Scheduling in Cloud Computing Environment", *Future Generation Computing Systems*, Vol. 56, pp. 640-650, 2016.

[24] G. Reddy N. Reddy, and S. Phanikumar, "Multi Objective Task Scheduling Using Modified Ant Colony Optimization in Cloud Computing", *International Journal of Intelligent Engineering and Systems*, Vol. 11, No. 3, pp.242-250, 2018.

[25] P. Krishnadoss and P. Jacob, "OCSA: Task Scheduling Algorithm in Cloud Computing Environment", *International Journal of Intelligent Engineering and Systems*, Vol. 11, No. 3, pp.271-279, 2018.

[26] A. B. A. Muthu and S. Enoch, "Optimized Scheduling and Resource Allocation Using

Evolutionary Algorithms in Cloud Environment", *International Journal of Intelligent Engineering and Systems*, Vol. 10, No. 5, pp.125-133, 2017.

[27] M. LawanyaShri, S. Subha, and B. Balusamy, "Energy-Aware Fruit-fly Optimisation Algorithm for Load Balancing in Cloud Computing Environments", *International Journal of Intelligent Engineering and Systems*, Vol. 10, No. 1, pp.75-85, 2017.

[28] M. Geetha and G. K. Mohan, "Metaheuristic Based Virtual Machine Task Migration Technique for Load Balancing in Cloud", In: *Krishna A., Srikantaiah K., Naveena C. (eds) Integrated Intelligent Computing, Communication and Security. Studies in Computational Intelligence,* Vol. 771, Springer, 2019.

[29] A. Varasteh and M. Goudarzi, "Server Consolidation Techniques in Virtualized Data Centers: A Survey", *IEEE Systems Journal*, Vol. 11, No. 2, 2017.

[30] C. Lin, P. Liu, and J. Wu, "Energy-efficient Virtual Machine Provision Algorithms for Cloud Systems", In: *Proc. of the Fourth IEEE International Conference Utility and Cloud Computing*, 2011.

[31] S. Takeda and T. Takemura, "A Rank based VM Consolidation Method for Power Saving in Datacenters", *IPSJ Online Transactions*, 2010.

[32] L. Hu, H. Jin, X. Xiong, and H. Liu, "Magnet, A Novel Scheduling Policy for Power Reduction in Cluster with Virtual Machines", In*: Proc. of the IEEE International Conference on Cluster Computing*, 2008

[33] L. Liang, W. Hao, L. Xue, J. Xing, H. W. Bo, W. Q. Bo, and C. Ying, "Green Cloud: A New Architecture for Green Data Center", In*: Proc. of the Sixth International Conference Industry Session on Automatic Computing and Communication Industry Session*, 2009.

[34] J. Yang, P. Liu, and J. Wu, "Workload Characteristics-Aware Virtual Machine Consolidation Algorithms", In*: Proc. of the Fourth International Conference on Cloud Computing Technology and Science*, 2012.

[35] K. Gupta and V. Katiyar, "Energy Aware Virtual Machine Migration Techniques for Cloud Environment", *International Journal of Computer Applications*, 2016.

[36] G. Pragya and Manjeey, "A Review on Energy Efficient Techniques in Green Cloud Computing", *International Journals of Advanced Research in Computer Science and Software Engineering,* Vol. 5, 2015.

[37] A. Banerjee, P. Agrawal, and N. S. N. Iyengar, "Energy Efficient Model for Cloud Computing", *International Journal of Energy Information and Communications,* Vol. 4, Issue 6 , 2013.

[38] K. Gupta and V. Katiyar, "Energy Aware Scheduling Framework for Resource Allocation in a Virtualized Cloud Data Centre", *International Journal of Engineering and Technology*, 2017.