Hybrid Model for Load Balancing and Server Consolidation in Cloud

Geetha Megharaj¹* Mohan Kabadi²

¹Department of Computer Science and Engineering, Sri Krishna Institute of Technology, Bangalore, Karnataka, India
²Department 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.
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 relieved 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

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—
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, $c_i$ indicates the number of computing node in $VM_y$, $m_i$ indicates the memory capacity of $VM_y$, $t_i$ indicates the $i^{th}$ task present inside $VM_y$, $c_i$ is the CPU utilization ratio of $t_i$, if $t_i$ is running on multiple CPUs, then, $c_i$ is the sum of CPU utilization ratio for every CPU on which $t_i$ is being executed, $m_i$ is the memory utilization ratio of $t_i$ and $p_i$ represents the power consumption of $t_i$, which is represented in Eq. (1)

$$p_i = c_i m_i$$ (1)

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

$$p = \frac{\sum_{i=1}^{n_y} p_i}{c}$$ (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)

$$Overloaded(VM_y) = \begin{cases} 1, \text{ if } T_c \leq \frac{\sum_{i=1}^{n_y} p_i}{c} \\ 0, \text{ otherwise} \\ or \quad T_p \leq p \\ 1, \text{ otherwise} \end{cases}$$ (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 $i^{th}$ extra task of $VM_y$ indicated by $t_i$ is considered for migrating to $VM_z$. The benefit of this migration scenario is modeled through a score function represented in Eq. (4). Here, $scoreEff(t_i, VM_z)$ indicates the benefit score of the considered migration; lower the score, better will be the migration scenario; $exeET_i$ indicates the predicted execution time of $t_i$ when $t_i$ is migrated to $VM_z$, and this metric is represented in Eq. (5); $transfer(t_i, VM_z)$ indicates the cost of transferring $t_i$ to $VM_z$, and this metric is represented in Eq. (6); the size of execution data used by $t_i$ is indicated by $d_i$;
the available bandwidth for transferring data between \( VM_i \) and \( VM_j \) is indicated by \( bw_{ij} \).

\[
\text{score}_{ET}(t_{iy}, VM_x) = \text{exe}_{ET}(t_{iy}, VM_x) + \text{trans}_{f}(t_{iy}, VM_x)
\]

(4)

\[
\text{exe}_{ET}(t_{iy}, VM_x) = \frac{d_{iy}}{cz + mx}
\]

(5)

\[
\text{trans}_{f}(t_{iy}, VM_x) = \frac{d_{iy}}{bw_{yz}}
\]

(6)

After identifying the overloaded VMs, the set of extra tasks from these VMs indicated by \( t_{i1y1}, t_{i2y2}, \ldots, t_{iys} \) has to be migrated to suitable VM set. Consider a candidate solution indicated by \( S_i \), having the VM set indicated by \( [VM_{z1}, VM_{z2}, \ldots, VM_{zs}] \), which can provide feasible migration to these extra tasks. Here, \( t_{i1y1}(1 \leq j \leq s) \) is considered for migrating to \( VM_{j0} \), 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, \( \text{migration\_score}_{ET}(S_i) \) represents the migration score.

\[
\text{migration\_score}_{ET}(S_i) = \sum_{j=1}^{s} \text{score}_{ET}(t_{iyj}, VM_{zj})
\]

(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 \( S_0 \) is the optimal candidate solution. Let’s assume: \( S_o \neq S_i \) and \( S_0 > S_i \). Since, the migration scoring function belongs to the class of monotonically non-decreasing functions, \( S_o \) cannot be lesser than \( S_i \) 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, \( \{r \leq s\} \) tasks have to be migrated to \( VM_{z1} \), and \( s-r \) tasks have to be migrated to \( VM_{z2} \). 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 \( > (\frac{s}{r}) \), 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, \( \text{task\_completion}(t_{iy}) \) indicates the task completion ratio of \( t_{iy} \), \( d_{iy} \) is the size of data used by \( t_{iy} \) and \( d_{iy} \) indicates the size of data already consumed by \( t_{iy} \).

\[
\text{task\_completion}(t_{iy}) = \frac{d_{iy}}{d_{iy}}
\]

(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.

\[
\text{task\_completion}(t_{iy}) \leq T_o
\]

(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)

\[
\text{task\_selected\_for\_migration} = \max_{t_{iy}} (c_{iy} + p_{iy})
\]

(10)

Suppose that, \( t_{iy} \) has to be migrated from \( VM_i \) and \( VM_i \) 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 \( exe_{iz}, trans_{f}(t_{iy}, VM_i), p_{iz}, g(T_{cc}, t_{iy}) \) and \( g(T_{pc}, t_{iy}) \) are represented in Eqs. (12), (13), (14), and (15)

respectively. Here, \( \text{score}(t_i, VM_j) \) indicates the migration score, \( \text{exe}_{iz} \) indicates the cost of executing ticks in VM, \( \text{transfer}(t_i, VM_j) \) indicates the transfer cost of transferring \( t_i \) from VM to VM, \( bw_{czi} \) indicates the bandwidth between VM and VM, \( p_i \) is the power consumed by VM when task \( t_i \) is migrated to VM. The functions \( g(T_c, t_i) \) and \( g(T_p, t_i) \) ensure that, the migration of \( t_i \) from VM to VM does not cause CPU utilization threshold and power consumption threshold violations.

Consider the situation where the set of tasks \( \{t_{1:iy1}, t_{2:iy2} \ldots t_{isy} \} \) 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, \( t_{ij1} \) will be migrated to \( VM_{z1} \), \( t_{ij2} \) will be migrated to \( VM_{z2} \) and so on \( t_{isy} \) will be migrated to \( VM_{zs} \). There is no restriction that, the VMs in the candidate solution set should be distinct. The migration score for this candidate solution is indicated by migration \( \text{score}_{RT}(S_2) \) is represented in Eq. (16). Here, \( t_{ij} \rightarrow VM_{zj} \) \( (1\leq j \leq s) \) indicates that the task \( t_{ij} \) has already been assigned to \( VM_{zj} \) and is being executed inside it. The CPU and memory utilization ratio of \( t_{ij} \) in \( VM_{zj} \) is assumed to be same as observed when \( t_{ij} \) was executing inside \( VM_{zi} \). The operator | is interpreted as such that.

\[
\begin{align*}
\text{score}_{RT}(t_{iy}, VM_z) &= \text{exe}_{RT_{iz}} + \text{transfer}(t_{iy}, VM_z) + \hat{p}_z - (g(T_c, t_{iy}) + g(T_p, t_{iy})) \\
\text{exe}_{RT_{iz}} &= \frac{dy}{c_z \cdot x_{cy} + m_{zy}} \\
p_z &= p_z + \frac{py}{cz} \\
g(T_c, t_{iy}) &= \\
&= \begin{cases} \\
T_c - \frac{\sum z_i c_j + z_i c_y}{cz}, & \text{if } T_c - \frac{\sum z_i c_j + z_i c_y}{cz} > 0 \\
-\infty, & \text{if } T_c - \frac{\sum z_i c_j + z_i c_y}{cz} \leq 0 
\end{cases} \\
g(T_p, t_{iy}) &= \\
&= \begin{cases} \\
T_p - (p_z + \frac{py}{cz}), & \text{if } T_p - (p_z + \frac{py}{cz}) > 0 \\
-\infty, & \text{if } T_p - (p_z + \frac{py}{cz}) \leq 0 
\end{cases}
\end{align*}
\]

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, \( PM_i \) indicate the \( k \)th PM in the CC, \( \text{num}(PM_k) \) indicate the number of active VMs in PM. Each PM is defined with a corresponding threshold indicated by \( SD(PM_i) \), 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, \( \text{shutdown}(PM_k) = 1 \) indicates that, PM should be shutdown, and \( \text{shutdown}(PM_k) = 0 \) indicates that, PM should be kept active.

\[
\text{shutdown}(PM_k) = \begin{cases} \\
1, & \text{if } \text{num}(PM_k) < SD(PM_k) \\
0, & \text{otherwise} \end{cases}
\]

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

Let, \( t_i \) indicate the \( i \)th extra task submitted to VM, \( PM \in \tilde{SD} \), and suppose it can be migrated to VM, which is hosted in that PM \( \notin \tilde{SD} \). The migration of \( t_i \) also requires the migration of data associated with \( t_i \). The merit of this migration is analyzed through a scoring function represented in Eq. (18). Here, \( \text{score}(t_i, VM) \) indicates the score of migration strategy which migrates \( t_i \) from VM to VM, \( \text{exe}_{iz} \) indicates the estimated execution time of \( t_i \) inside VM, which is represented in Eq. (19).
scoreSC\( (t_{ij}, VM_z) = exeSC_{iz} + transfer(t_{ij}, VM_z) \) \hspace{1cm} (18)

\[
exeSC_{iz} = \frac{d_{iy}}{c_z + m_z}
\] \hspace{1cm} (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 \( \hat{V}{\hat{M}} \) are batched together for migration. Consider the scenario, where the batch of extra tasks \([t_{1y1}...t_{2y2} ... t_{sys}] \) submitted to \( \hat{V}{\hat{M}} \) need to be migrated. Suppose, \([VM_{i1}, VM_{i2},...VM_{iz}] \) is a candidate solution for the required migration of tasks, wherein, \( t_{ijy} \ (1 \leq j \leq s) \) is considered to be migrated from \( VM_{ij} \) to \( VM_{iz} \) and this candidate solution is denoted as \( S_c \). Also, there is no restriction that, the VMs in the candidate solution should be distinct. The score of this migration scheme indicated by \( migration\_scoreSC(S_c) \) is represented in Eq. (20).

\[
migration\_scoreSC(S_c) = \sum_{i=1}^{s} scoreSC(t_{ij}, VM_z)
\] \hspace{1cm} (20)

Theorem 5. Minimizing the migration score function represented in Eq.(20) provides the optimal candidate solution for SC problem involving VM task migration.

\textbf{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.

\textbf{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.

\[
S = [S_1, S_2, S_3]^T
\] \hspace{1cm} (21)

The merit of the CSV \( S \) is analyzed through the scoring function indicated by \( hybrid\_migration\_score(S) \) represented in Eq. (22).

\[
hybrid\_migration\_score(S) = migration\_scoreET(S_1) + migration\_scoreRT(S_2) + migration\_scoreSC(S_3)
\] \hspace{1cm} (22)

Theorem 7. Minimizing the migration score function represented in Eq.(22) provides the optimal candidate solution for hybrid model problem.

\textbf{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.

\textbf{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.
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 i\textsuperscript{th} 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, \(\vec{V}_i(t+1)\) denotes the velocity of the i\textsuperscript{th} 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 \(\vec{X}_{\text{pbest}}\) indicate the global best solution obtained by all the particles until the current iteration respectively, W indicates a control variable, and \(r_1, r_2 \in [0, 1]\) are the random factors.

\[
\vec{X}_i(t+1) = \vec{X}_i(t) + \vec{V}_i(t+1) \\
\vec{V}_i(t+1) = W\vec{V}_i(t) + D_1 r_1 (\vec{X}_{\text{pbest}} - \vec{X}_i(t)) + D_2 r_2 (\vec{X}_{\text{gbest}} - \vec{X}_i(t))
\]

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 \(P = \{p_1, p_2, \ldots, p_r\}\) 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 \(\vec{X}_{\text{pbest}}\) are calculated through local\_best(score)\(_i\) and global\_best(P, \(\vec{X}_{\text{pbest}}\)) respectively. The particles continue to search until the acceptable solution is found, and which is calculated through acceptable(\(\vec{X}_{\text{gbest}}\)).

Algorithm 1 PSO Algorithm for VM task migration

\[
P = \{p_1, p_2, \ldots, p_r\} \\
\text{initialize\_PSO}(P) \\
t = 0 \\
\text{While flag = 0 do} \\
\text{Do for } i=1 \text{ to } r \text{ do} \\
\text{score}_i = \text{compute\_score}(\vec{X}_i(t)) \\
\text{\(\vec{X}_{\text{pbest}} = \text{local\_best(score)}_i\)} \\
\text{\(\vec{X}_{\text{gbest}} = \text{global\_best}(P, \vec{X}_{\text{pbest}})\)} \\
\text{If acceptable(\(\vec{X}_{\text{gbest}}\)) then} \\
\text{flag = 1} \\
\text{end if} \\
\text{end for} \\
\text{t = t + 1} \\
\text{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
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.

\[
APUR = \frac{\Sigma_{VM_{inCC}} P_Y}{|CC|} \tag{25}
\]

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.

\[
AVEXE = \frac{\Sigma_{VM_{inCC}} \text{execTime}}{|CC|} \tag{26}
\]
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, $U(d_{iy}) = 1$.

$$TCOST = \sum_{y} \sum_{i} d_{iy} U(d_{iy})$$  \hspace{1cm} (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, $\text{max}(exeRT_{xy})$ indicates the execution time of $t_{xy}$, which has the highest execution latency in $VM_y$. Also, $exeRT_{xy}$ is expressed in terms of hours.

$$ATPT = \frac{\sum_{y} TPT(VM_y)}{|CC|}$$  \hspace{1cm} (30)

$$TPT(VM_y) = \frac{n_y}{\text{max}(exeRT_{xy})}$$  \hspace{1cm} (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

---

**Table 1. Simulation parameter settings**

<table>
<thead>
<tr>
<th>Simulation Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of VMs considered</td>
<td>Varied between 1000- 5000</td>
</tr>
<tr>
<td>Number of computing nodes/CPU in each VM</td>
<td>Varied between 5 to 20</td>
</tr>
<tr>
<td>Main memory capacity for each VM</td>
<td>Varied 4GB/ 8GB/16GB</td>
</tr>
<tr>
<td>Number of tasks executing in each VM</td>
<td>Varied between 10 to 50 tasks</td>
</tr>
<tr>
<td>Bandwidth between any 2 VMs</td>
<td>Varied between 100mbps to 500mbps</td>
</tr>
<tr>
<td>CPU utilization ratio for any task</td>
<td>Varied between 0.02 to 0.8</td>
</tr>
<tr>
<td>Memory utilization of each task</td>
<td>Varied between 0.02 to 0.8</td>
</tr>
<tr>
<td>Number of PSO search particles</td>
<td>Varied between 5 – 25</td>
</tr>
<tr>
<td>Number of Computing nodes allotted for each PSO particle</td>
<td>1</td>
</tr>
<tr>
<td>Threshold $T_c$</td>
<td>0.7</td>
</tr>
<tr>
<td>Threshold $T_p$</td>
<td>0.6</td>
</tr>
<tr>
<td>Size of task data</td>
<td>Varied between 1GB to 10GB</td>
</tr>
<tr>
<td>Threshold $T_a$</td>
<td>Varied between 0.05 – 0.25</td>
</tr>
<tr>
<td>Number of VMs present in each PM indicated by $tvn(PM_k)$</td>
<td>Varied between 0 to 200 (randomized)</td>
</tr>
<tr>
<td>$nvm(PM_k)$</td>
<td>$0.5 \times tvn(PM_k)$</td>
</tr>
<tr>
<td>Number of extra tasks for a VM during le</td>
<td>Poisson distributed with $\lambda = 5$</td>
</tr>
<tr>
<td>$\text{min SD}(PM_k)$</td>
<td>Varied between $[5-25]$</td>
</tr>
<tr>
<td>Power consumed by each VM</td>
<td>Varied between 0 to 1(normalized)</td>
</tr>
</tbody>
</table>

$$AVEXE = \frac{\sum_{i \in CC} exeRT_{iz}}{|CC|}$$  \hspace{1cm} (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 $VM_y$.

$$ARUR = \frac{\sum_{VM_y \in CC} CPU_y}{|CC|}$$  \hspace{1cm} (27)

$$CPU_y = \frac{\sum_{i \in CC} CPU_y}{|CC|}$$  \hspace{1cm} (28)
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.

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.
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.
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 VMM 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


