A Multi-Objective Load Balancing System for Cloud Environments

Fahimeh Ramezani[a,1], Jie Lu[a], Javid Taheri[b], Albert Y. Zomaya[c]

[a] Decision Support and e-Service Intelligence Lab

Centre for Quantum Computation & Intelligent Systems

School of Software, Faculty of Engineering and Information Technology

University of Technology Sydney, NSW 2007, Australia

[b] Department of Computer Science, Karlstad University, Karlstad, Sweden

[c] Centre for Distributed and High Performance Computing, School of Information Technologies,

University of Sydney, NSW 2006, Australia

Abstract

Virtual Machine (VM) live migration has been applied to system load balancing in cloud environments for the purpose of minimizing VM downtime and maximizing resource utilization. However, the migration process is both time- and cost-consuming as it requires the transfer of large size files or memory pages and consumes a huge amount of power and memory for the origin and destination Physical Machine (PM), especially for storage VM migration. This process also leads to VM downtime or slowdown. To deal with these shortcomings, we develop a Multi-objective Load Balancing (MO-LB) system that avoids VM migration and achieves system load balancing by transferring extra workload from a set of VMs allocated on an overloaded PM to other compatible VMs in the cluster with greater capacity. To reduce the time factor even more and optimize load balancing over a cloud cluster, MO-LB contains a CPU Usage Prediction (CUP) sub-system. The CUP not only predicts the performance of the VMs but also determines a set of appropriate VMs with the potential to execute the extra workload imposed on the VMs of an overloaded PM. We also design a Multi-Objective Task Scheduling optimization model using Particle Swarm Optimization (MOTS-PSO) to migrate the extra workload to the compatible VMs. The proposed method is evaluated using a VMware-vSphere based private cloud in contrast to the VM migration technique applied by vMotion. The evaluation results show that the MO-LB system dramatically increases VM performance while reducing service response time, memory usage, job makespan, power consumption, and the time taken for the load balancing process.

Keywords: Cloud computing, Particle swarm optimization, Virtual machine migration, Task scheduling.

[1] Corresponding author, Tel: +61 9514450, Email addresses: Fahimeh.Ramezani@uts.edu.au, Jie.Lu@uts.edu.au, Javid.Taheri@kau.se, albert.zomaya@sydney.edu.au

1. Introduction

Cloud computing delivers scalable on-demand services include Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) over the Internet. A cloud provides the IaaS, PaaS, and/or SaaS through its own virtualized resources, which are created over its underlying physical resources. Typically, a cloud virtualized resource is a set of specification and configuration files called a Virtual Machine (VM) [1, 2].

Due to the dynamic nature of cloud environments, the workload of VMs fluctuates dynamically, leading to imbalanced loads and the utilization of virtual and physical cloud resources. VM migration is used in this situation to relax the workload by moving a VM from an overloaded Physical Machine (PM) to an under-loaded PM. In addition, VM migration is applied when IaaS customers ask to scale up their assigned VMs, while the original host PM has no idle resources available [3]. VM migration by the suspend/resume strategy or live migration is the process of copying the complete state of a VM from one PM to another for stronger computation power, larger memory, fast communication capability, or energy saving [4]. In suspend/resume VM migration, the execution of the VM is suspended during the migration process; applications are halted until the VM is fully migrated to the destination host [5-8]. VM downtime in this method equals the VM migration time plus the time taken to suspend and resume the VM. Using the stop-and-copy process in live migration, the running instance of a VM is migrated between hosts. VM downtime is thus much less in stop-and-copy (~10-120 seconds depending on the load) than suspend/resume (~180-600 seconds depending on the load).

We believe that if a VM has small size, it is reasonable to migrate the VM to a new physical host. However, when the VM is large, VM migration is not the optimal solution. The live VM migration process results in dirty memory as a result of the pre-copy process, utilizes a large amount of memory in the primary PM and new host PM, causes the VM to slow down during the migration process, carries the risk of losing last customer activities, and is cost- and time-consuming. The resume/suspend migration strategy not only has live migration shortcomings but also causes lengthy VM downtime.

In this study, a Multi-Objective Load Balancing (MO-LB) system is developed to eliminate the need for VM migration to solve the problem of an over-utilized PM, and to scale up a VM that is located on a PM with no available resources. To do this, the MO-LB system reduces the workload of a set of VMs — that deliver SaaS or PaaS and are located on an over-utilized PM— and transfers their extra workload to a set of compatible VM instances located on underutilized PMs. A compatible VM is a VM with the same

OS as the primary VM, and has the required application/software stack to execute the scheduled tasks. Cloud providers need to solve such problems as this in the shortest possible time to satisfy the Service Level Agreement (SLA). Therefore, a prediction sub-system is developed to predict CPU usage by VMs which is able to predict PM hotspots before low performance occurs. It also creates the opportunity to predict a set of compatible under-loaded VMs that can execute the workload that has accumulated in the task queues of a set of primary VMs allocated on a possible over-utilized PM. This helps the hypervisor layer to make an accurate decision and solve the problem before it arises. We also develop a multi-objective optimization sub-system for transferring these accumulated tasks from the primary VMs to the destination VMs with minimum task transfer time, task execution cost/time, power consumption, and task queue length.

In summary, the contribution of this paper is to develop an MO-LB system that eliminates VM migration to achieve system load balancing through the creation of two sub-systems: (1) a CPU usage prediction sub-system, and (2) a Multi-Objective Task Scheduling sub-system. The efficiency of the proposed solution is evaluated by comparison with vMotion in both live and storage VM migration using a VMware-vSphere based private cloud and HTCondor functionality. The evaluation parameters are makespan, execution time, memory usage, and total time taken for load balancing.

The rest of this paper is organized as follows. Works related to load balancing methods are described in Section 2. In Section 3, a conceptual model and the main algorithm for the MO-LB system are proposed. This model is completed by the developed VM workload prediction and the multi-objective task scheduling optimizing sub-systems presented in Section 4 and Section 5 respectively. The model is evaluated in Section 6. Lastly, the conclusion and future works are explained in Section 7.

2. Related Works on Load Balancing in Cloud Environments Using VM Migration Techniques

Virtualization technique has improved utilization and system load balancing by enabling database [9-12] and VM migration, which has resulted in significant benefits for cloud computing [13]. Several methods have been proposed to migrate a VM from one physical host to another with more available resources for optimizing cloud utilization. These methods are categorized in two main classes: (1) suspend /resume strategy, and (2) live migration.

The suspend/resume VM migration approach has three steps: pause the original VM, copy the VM's related data (memory pages and processor state) to a new host PM, and then resume the VM on the desti-

nation host [5-7]. Using this method, applications running on the VM need to be stopped and are not made available until the migration process has been completed and all the data have been transferred to the new destination. Moreover, this method results in long VM downtime. To reduce downtime, the ZAP system [14] only transfers a process group, but it still uses a stop-and-copy strategy. In contrast, live migration, in which a running instance of VM is migrated between hosts in a local area network, eliminates the stop-and-copy process and minimizes VM downtime. Jun and Xiaowei [15] developed a VM live migration policy for the IPv6 network environment. In this migration method, the VM does not provide new services but continues its work and then stops after completing its old services. A pre-copy migration method is applied by the vMotion component of VMware vSphere in [16], and Xen hypervisor in [17] for live migration. Using this method, VM's run-time memory state files are pre-copied (migrated) from the source host to the destination host while the VM is still working. This method generates a huge amount of dirty memory and takes a long time, because it is necessary to transfer a large amount of data. In addition, the dirty memory generation rate in some cases is faster than the pre-copy speed, in which case live migration will be prolonged. To overcome these drawbacks, Jin et al. [4] suggested using the pre-copy based model from VM live migration in combination with an optimized algorithm that reduces the speed of changing memory by controlling the CPU scheduler of the VM monitor. To ease the VM migration process, Nicolae et al. [18] developed a repository check pointing strategy called BlobCR that frequently stores live snapshots of the entire VM instances disk. There are also several VM live migration techniques that consider power consumption reduction as well as downtime and migration time. Liao et al. [19] developed a live VM mapping framework to map VMs onto a set of PMs without significant system performance degradation, while at the same time reducing power consumption. Sallam and Li [3] also suggested a multi-objective VM migration technique that considers power and memory consumption, thus making live VM migration more beneficial for cloud providers.

Lin et al. [20] believed that load balancing strategies that focus on VM migration for optimizing on-demand resource provisioning needed to be improved. They proposed a threshold-based dynamic resource allocation approach for load balancing in the cloud environment that dynamically allocates the VMs among the cloud's applications based on their load changes. Atif and Strazdins [21] also developed a similar cloud utilization optimization framework for Application as a Service (AaaS). They used virtual machine monitor facilities (which have traditionally been used for live migration) to create sets of homogenous clusters of computing frames (VMs). They used these clusters to schedule or migrate application tasks

over a set of homogenous VMs based on estimated task execution time to optimize resource utilization and enhance application performance. However, this method cannot be used when a determined homogenous cluster has high utilization and is in an overloaded state. The reviewed literature related to this study is summarized in Table 1.

The fundamental drawback of these load balancing approaches is that the majority attempt to migrate the VM [4, 15, 17, 19]. In our previous work [22], we proposed a Task-Based System Load Balancing using a Particle Swarm Optimization (TBSLB-PSO) conceptual model to overcome these shortcomings. The TBSLB-PSO achieves system load balancing by migrating tasks from primary VMs on an overloaded PM to a set of compatible VM instances located on under-loaded PMs, instead of migrating VMs in their entirety. However, although VM migration has been eliminated by TBSLB-PSO, this model is not applicable to VMs that are delivered as IaaS because cloud providers do not have access to the applications running on these VMs.

Using the MO-LB system, not only is the need for VM migration eliminated: the system is also applicable to VMs that deliver different types of cloud services. A CPU utilization prediction sub-system is also added to the previous model to determine primary VMs, PM hotspots and new destination VMs, for the purpose of executing tasks located in the task queues of the primary VMs. The MOTS-PSO subsystem of MO-LB is also improved so that it is compatible with real cloud environments and has four objective functions, while the TBSLB-PSO is not applicable in real cloud environments and has two objective functions. The MO-LB system reduces the amount of dirty memory produced, as well as the consumption of load balancing time and power, compared to VM migration. Furthermore, the proposed system is not restricted to distributing the extra workload over a set of VMs in predefined clusters, because the new destination VMs are determined dynamically over the entire local cloud environment.

Table 1. Summary of reviewed literature related to load balancing in cloud environments

References	Key Development
1- Suspend/resume VM migration	
ZAP system in [14]	Applied suspend/resume VM migration strategy that only transfers a process group
2- Live VM migration	
Jun and Xiaowei [15]	Developed a VM live migration policy for the IPv6 network environment
vMotion in [16] and Xen in [17]	Applied a pre-copy migration method for live migration
Jin et al. [4]	Combined the pre-copy based model and an optimized algorithm that reduces the speed of

	changing memory
Nicolae et al. [18]	Developed a repository check pointing strategy called BlobCR for live migration
Liao et al. [19]	Developed a live VM mapping framework with minimum system performance degradation while reducing power consumption
Sallam and Li [3]	Developed a bi-objective VM migration technique that considers power and memory consumption
3- Other load balancing techniques	
Lin et al. [20]	Developed a threshold-based dynamic resource allocation approach for load balancing
Atif and Strazdins [21]	Developed a cloud utilization optimization framework for AaaS to optimize load balancing

3. The Multi-Objective Load Balancing System

VM migration (live and storage migration) is applied by a hypervisor (or Virtual Machine Monitor (VMM)) such as VMware EXSi to manage cloud resources and balance the load over PMs [4, 15, 17, 19]. VM migration has important applications in dynamic resource management for cloud-based systems and large data centers. In these environments, a group of VMs can begin to compete for resources provided by a single PM. A hotspot occurs when the performance of VMs degrades because the PM is unable to respond to the resource demand. The opposite situation, when resources on a PM are underutilized, is termed a cold spot. This happens when the running VMs consume only a tiny fraction of the resources provided by the hosting PM. Both hotspots and cold spots can be handled by moving one or more VMs from an overloaded PM to another PM with available physical resources. In hotspot mitigation, the selected VMs are moved to a less loaded PM. Server consolidation is the strategy for handling cold spots by regrouping VMs from lightly-loaded hosts to a smaller subset of PMs, thus freeing up the remaining PMs for resource-hungry VMs [23, 24]. VM migration is also applied to scale up VMs that are delivered as IaaS based on customer demand when the original host PM has no idle resource availability [3].

Cloud providers benefit from VM migration for small VMs as the entire process is completed in seconds. However, VM migration for large VMs results in dirty memory, utilizes a large amount of memory in the primary PM and destination PM, causes the VM to slow down during the migration process, and carries the risk of losing last customer activities. Therefore, the need for a replacement solution that can achieve higher cloud utilization, increase the performance of primary VMs, and as a result allow cloud providers to deliver higher QoS at lower cost has been recognized.

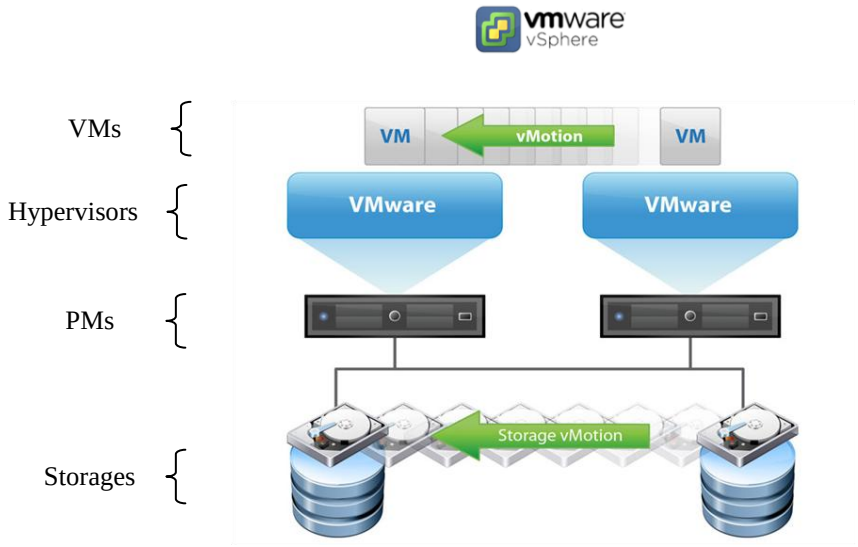


Figure 1: VM migration using vMotion

The main goal of this study is to find a way to reduce the need for VM migration, especially for large VMs in two particular situations: (1) the primary PM is over-utilized/overloaded; (2) a VM that is delivered as IaaS needs to be scaled up and there is no available capacity on the host PM.

The first step is to create clusters of PMs with a set of VMs that are applied to deliver PaaS or IaaS, and another set of VMs to deliver AaaS/SaaS, in which cloud service providers distribute jobs/DAGs (Directed Acyclic Graph) of applications. Instead of conducting VM migration when the host PM becomes overloaded, the solution is to reduce the load of the set of VMs that deliver AaaS/SaaS, or stop them from working. Their extra workload is then allocated to compatible VMs that are allocated on the other PM and have the resources available to execute the workload. This solution is similar to job/task scheduling, which is widely applied in cloud and grid environments. Using this solution, the load of the PM is reduced and there is no need to migrate some of its allocated VMs (see Figure 2). In the second situation where VM migration is required, the same solution is suggested. Then, the resources allocated to the VMs that are no longer working or have fewer jobs to do are allocated to the VM that needs more resources. Therefore, this VM can be scaled up and does not need to be migrated (see Figure 3).

HTCondor [25] and Pegasus-WMS [26] are two examples of systems that can be used to implement this idea. In both systems, a central management system periodically polls the status of all its "workers". Workers, in turn, are designed to collect and report the latest status (e.g., the number of available CPUs and amount of available memory) of the machine/VM in which they reside. This work was empirically tested using HTCondor v8.2. The central management system can also "kill" and "reinitiate" jobs should

it need to empty a higher-capacity VM to accommodate another job with greater requirements. Regardless of this capability, MO-LB implicitly assigns jobs to the "right" size VM so that it is not necessary to later reassign jobs. Because VMs are provided by a cloud infrastructure, we can also assume that MO-LB will spin-up larger size VMs as needed, or at least, replace several small VMs with a larger VM by destroying and/or resizing existing VMs.

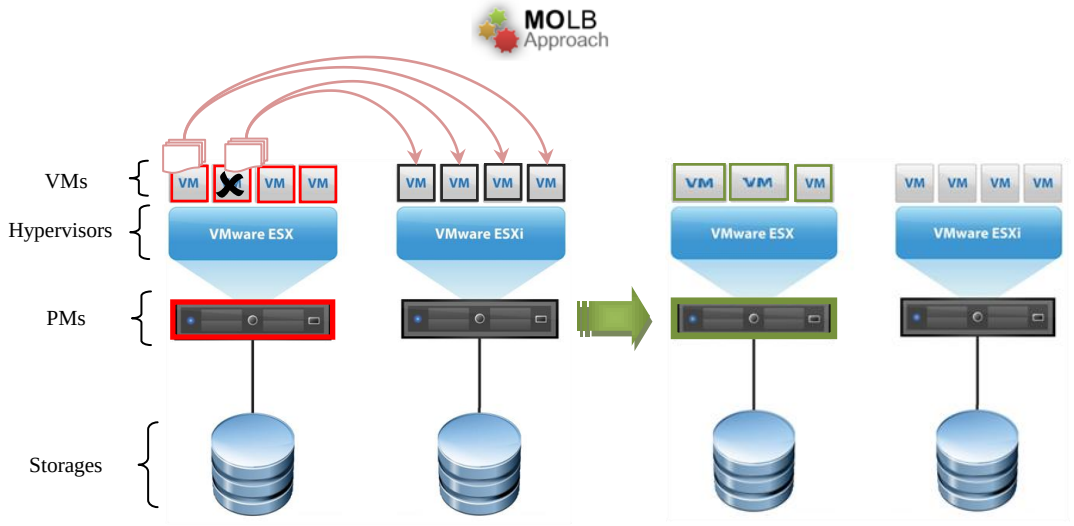


Figure 2: MO-LB solution for resolving the problem of an overloaded PM

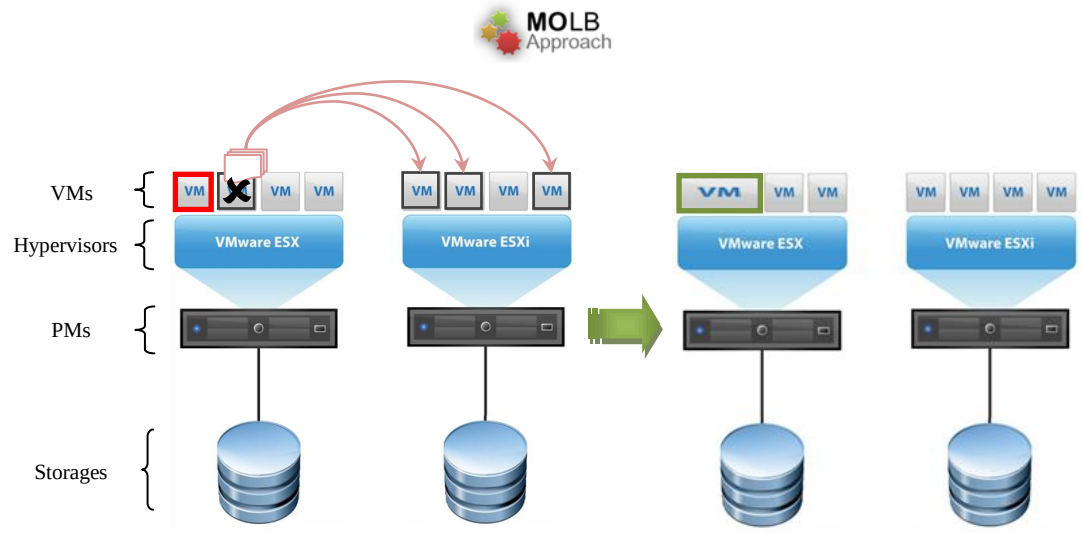


Figure 3: MO-LB to scale up a VM that delivers IaaS

The MO-LB system has four main parts: (1) Global Blackboard (GB), (2) CPU Usage Prediction (CUP) sub-system, (3) Multi-Objective Task Scheduling sub-system applying PSO (MOTS-PSO sub-system),

and (4) Decision Maker Centre (DMC). The first three parts communicate with each other and report to the DMC of the MO-LB system to conduct load balancing. An overview of the MO-LB system is given in Figure 4.

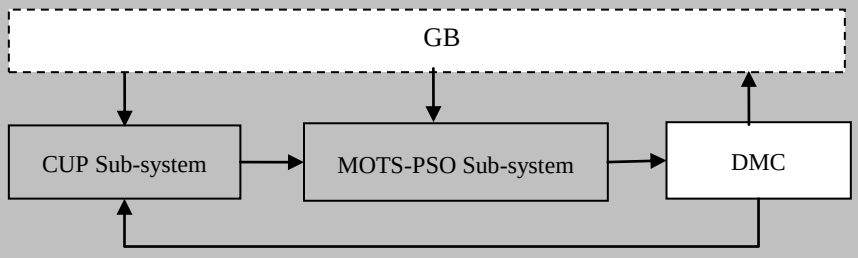


Figure 4: Overview of the MO-LB system

The first element is the GB on which all Virtual Machine Monitors (VMMs) and task schedulers share their data and information about VM features, jobs and scheduled tasks in a cluster. This data includes the number of CPUs, free memory and bandwidth allocated to VMs. Additional information about the VMs provided for SaaS and PaaS includes the number of tasks to be executed, the task execution time, and the resources required for the tasks (number of processors required, CPU and memory usage). In addition, this blackboard contains online information about PMs (physical resources), such as the number and speed of their processors (CPUs), the amount of free memory and hard disk they have, their situation (idle or active), and their associated VMM. The information about SLA constraints is also gathered on this blackboard to monitor QoS criteria. The stored variables on GB are summarized in Table 2.

The second element of the MO-LB framework is the CUP sub-system which predicts the two situations that are targeted in this study. Based on the predicted situations, the DMC determines whether the MO-LB system or VM migration should be applied to solve the problem. If the VMs on an overloaded PM are small, the DMC suggests VM migration, otherwise, the MO-LB system is applied. In the first situation, where the primary PM has become overloaded, the CUP sub-system applies blackboard data and information related to the VMs' CPU usage to predict which VMs are likely to reach maximum utilization and exhibit low performance, thereby predicting hotspots in the PM. Based on the prediction result of the CUP sub-system, some VMs on the overloaded PM that deliver SaaS/PaaS are determined as primary VMs to reduce their workload. In the second situation where a VM that is delivered as IaaS needs to be scaled up, the CUP determines a set of VMs that deliver SaaS/PaaS in the neighborhood of the targeted

VM as the set of primary VMs. This method also determines a set of compatible high performance VMs that can be used to execute the extra workload imposed on the set of primary VMs.

The MOTS-PSO sub-system is developed as the third main element of the MO-LB framework. The MOTS-PSO sub-system applies all the information supplied by the other two parts of the framework (GB and CUP) to determine the optimal pattern for scheduling extra tasks from the primary VMs to a set of compatible destination VMs. The MOTS-PSO sub-system considers the minimization of task transfer time, task execution cost, power consumption in the corresponding data center, and task queue length in the destination VMs to find the optimal solution. The DMC of the MO-LB system then transfers those tasks and their corresponding data to the pre-determined set of VMs based on the optimal suggested task scheduling pattern.

The main algorithm of MO-LB is summarized as Algorithm 1. The proposed MO-LB framework illustrated in Figure 5 shows how MO-LB sub-systems communicate with each other and perform the steps of the MO-LB algorithm. The CUP and MOTS-PSO sub-systems referred to above are described in detail in Section 4 and Section 5 respectively.

Algorithm1. The MO-LB main algorithm

Input: All variables on the global blackboard (Summarized in Table 2).

[Begin MO-LB algorithm]

1. Monitor MO-LB blackboard data to collect VM information including: VM tasks, memory and CPU usage, the amount of virtual resources (CPU and memory), etc.

2. Predict VM CPU usage by applying CUP sub-system to determine primary over-utilized VMs on an overloaded PM, the PM host spots, and destination under-loaded VMs (**Algorithm 2**).

3. If VM is small and its host PM is also overloaded, then migrate it and go to Step 7.

Else

4. Predict a set of compatible VMs as the new possible destinations for the determined tasks by applying CUP sub-system (**Algorithm 2**).

5. Determine set of tasks that have accumulated in the task queue of each primary VM.

6. Determine the optimal task scheduling pattern to reschedule tasks onto new determined destination VMs by applying MOTS-PSO sub-system (**Algorithms 3**), and transfer tasks and their corresponding data to the determined VMs.

7. Update the blackboard and scheduler information to include the current properties of PMs and VMs.

[End MO-LB algorithm]

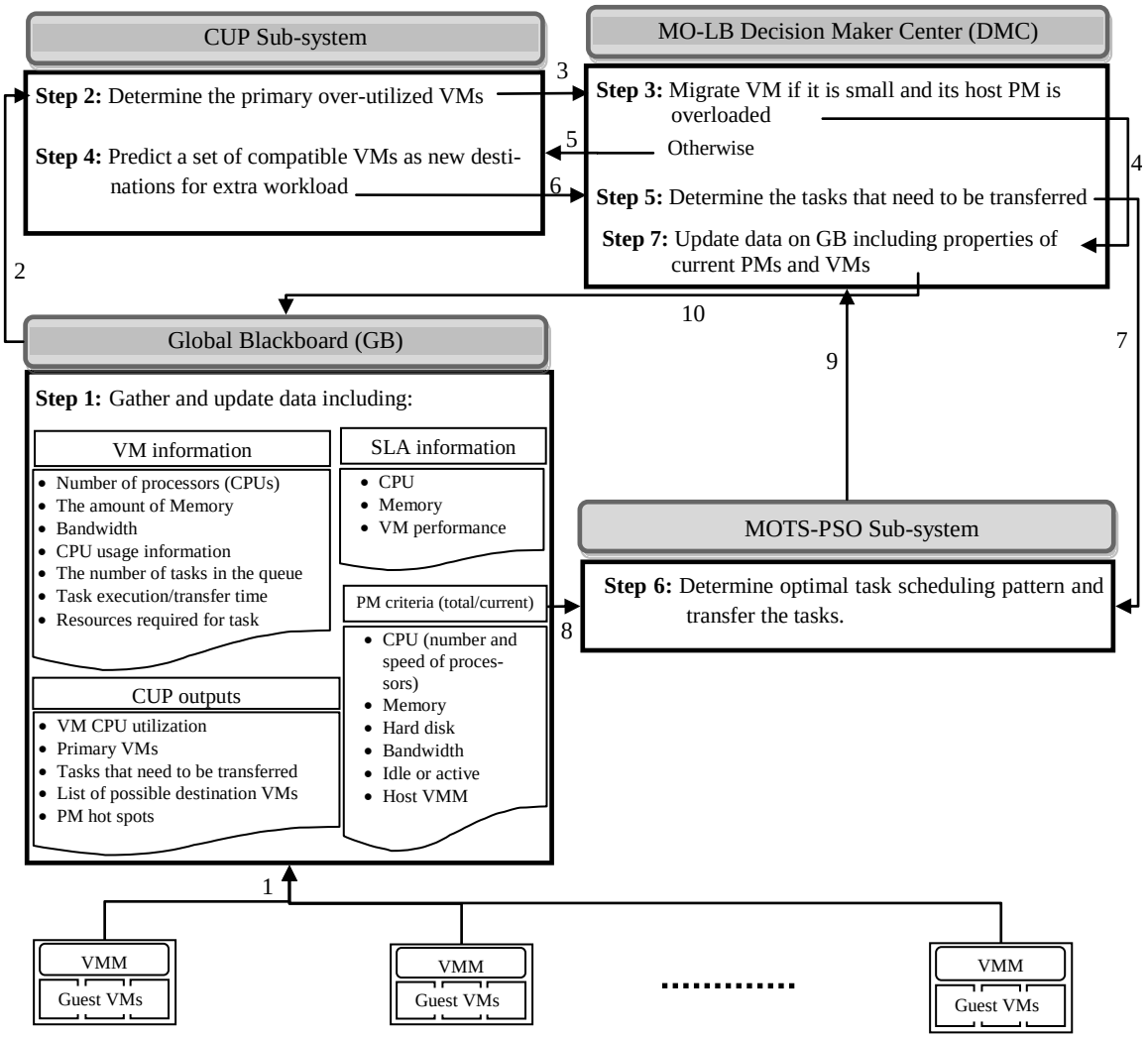


Figure 5: The MO-LB framework

4. CPU Usage Prediction Sub-System

Most researches in the area of VM workload prediction have applied prediction methods such as neural networks, pattern recognition and linear regression to forecast the workload of VMs or their CPU usage in the cloud environment [27]. These methods predict the future workload of VMs by applying their previous workload patterns in time slot *t*, determined on the basis of related historical data [28]. We also apply linear regression in the developed CPU Usage Prediction (CUP) sub-system to predict CPU utilization patterns by VMs that are allocated on a PM, using historical data.

Considering the fluctuation in CPU usage and the fact that it might increase suddenly and decrease soon after, CPU usage should be checked frequently at specific times to estimate a VM's upcoming CPU usage and workload [27]. Therefore, we monitor CPU usage trends and fluctuations over a small period of

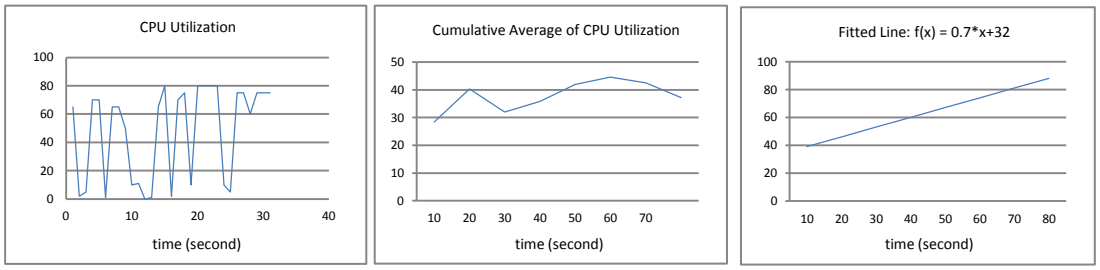
time (e.g., every two minutes) to forecast the VM's workload level for the next interval. The CUP is run every two minutes because the applied hypervisor is set to balance the load over PMs every five minutes. This time can be changed by the cloud provider, based on the primary settings of the hypervisor.

The CPU usage usually fluctuates dramatically and it is difficult to estimate its overall increasing or decreasing trend (see Figure 6a). Therefore, the cumulative average of CPU usage that is calculated every 20 seconds is used to estimate its trend during time slot Ts as follows:

$$CA^{k}_{cpu}(x) = \sum_{i=1}^{x*20} \frac{VM^{k}_{Ucpu}(i)}{x*20}, x \in \left\{1,2,3,..,div(\frac{Ts}{20})\right\} \quad (1)$$

Where variables Ts and i are in seconds, VM^{k}_{Ucpu} is the total amount of CPU utilization of VM_k and div is the integer division. Then, the corresponding continuous chart to $CA^{k}_{cpu}(x)$ is produced (see Figure 6b). The polynomial fitting tool in MATLAB is then applied to determine the overall trend of $CA^{k}_{cpu}(x)$ as shown in Figure 6c. $CA^{k}_{cpu}(x)$ has an increasing trend if the derivative of its fitted line ($fCA^{k}_{cpu}(x)$) is positive in Ts:

$$fCA'^{k}_{cpu}(x) \geq 0\ , x \in \left\{1,2,3,..,div(\frac{Ts}{20})\right\} \quad (2)$$



a **b** **c**

Figure 6: Estimating **the** CPU utilization trend in time slot *Ts*

However, the $CA^{k}_{cpu}(x)$ can be increasing while the CPU utilization of the VM is low. This study therefore assumes that this VM will be overloaded if the cumulative average of the CPU usage of VM_k ($CA^{k}_{cpu}(x)$) during time slot Ts has an increasing trend (see Equation 2) and the CPU utilization of the VM at the end of the time slot exceeds 80% i.e:

$$VM^{k}_{Ucpu}(ct) \geq 80\% \quad (3)$$

where *ct* is the current time, *t* is a given period of time (e.g., two minutes), and time slot Ts is the number of seconds between $\{ct - t, ct\}$. The CUP algorithm is summarized as follows.

Algorithm2. The CUP algorithm (Steps 2 and 4 of the MO-LB algorithm)

Input: All variables in Table 2.

[Begin CUP algorithm]

6.1. Monitor MO-LB blackboard data to calculate the value of the following variables:

- $VM^k_{Ucpu}(x), CA^k_{cpu}(x), fCA^k_{cpu}(x), fCA'^k_{cpu}(x)$ and $VM^k_{Ucpu}(ct)$

6.2. determine the following information based on CUP rules:

- The primary VMs as a result of their high CPU utilization
- The PMs that are at risk of becoming overloaded
- The primary set of VMs that is targeted to stop working or that has a reduced workload
- A set of compatible VMs as the new destination for the extra workload imposed on the primary VMs

[End CUP algorithm]

---

5. Multi-Objective Task Scheduling Optimization Sub-System

MOTS-PSO is designed as part of the MO-LB framework to find an optimal solution to the scheduling of tasks from primary VMs to a set of new destination VMs (Step 6 of the MO-LB algorithm). This optimization model has four conflicting objectives, namely: task transfer time, task execution cost/time, length of VM task queue, and power consumption. In our previous work [29], a four-objective optimization model that considers the same aspects of task scheduling was developed. The objective function formulas in [29] were designed for a simulation cloud environment (i.e. CloudSim [30]) and the model has been evaluated using the CloudSim toolkit. In this paper, however, we have changed the objective function formulas to make them compatible with real cloud infrastructures, such as the VMware-vSphere based clouds used in this work. This study focuses on scheduling Bag-of-Tasks (BoT) applications for SaaS. In BoT applications, the completion of one task does not affect the completion of other tasks, and only one task is executed on a computer processor (CPU) at a time. BoT applications are used for data mining, massive searches, parameter sweeps, simulations, fractal calculations, computational biology, and computer imaging [31, 32].

To formulate the MOTS-PSO objective functions, the following variables are defined:

Table 2. MOTS-PSO variables

Symbol	Definition
n	The number of tasks that have accumulated in the task queue of a primary VM (VM_l)
T_{set}^{l}	Set of tasks that has accumulated in the task queue of VM_l
DF_i	The task$_i$ file size (MB)
DO_i	The task$_i$ output file size (MB)
DI_i	The task$_i$ input file size (MB)
PR_i	$\{j \mid \text{task}_j$ is prerequisite for task$_i\}$
tm_i	The maximum level of memory required to execute task *i* (MB)
tc_i	The number of CPUs required to execute task *i*
tcu_i	The amount of CPU usage of task *i (GHz)*
$texe_k^i$	The total execution time of task *i* on VM_k(Hour)
$Texe_k$	The total task execution time on VM_k(Hour)
m	The number of VMs
VM_k	Virtual Machine *k*, *k*={1, 2, …, m}
VM_m^k	The amount of memory allocated to VM_k (MB ≈ 0.001 GB)
VM_c^k	The number of CPUs allocated to VM_k
VM_{ac}^k	The number of available CPUs allocated to VM_k
VM_{am}^k	The amount of available memory allocated to VM_k
N_{aCPU}^k	The number of active CPUs on VM_k
VM_{bw}^k	The bandwidth of VM_k (Mb/s)
$VM_{CPUSpeed}^k$	The CPU computing speed of VM_k *(GHz)*
RVM_m^k	The amount of available memory on VM_k
RVM_c^k	The number of available CPUs on VM_k
N_{PM}	The number of PMs in cloud
N_{aPM}	The number of active PMs in cloud
Svm_z	$\{k \mid VM_k \in zth\ \ PM, z \in \{1,2,\dots,N_{PM}\}\}$ = The set of indices of VMs located on zth PM
cp	The number of cloud providers
C_p	Maximum capacity of provider *p*
SP_p	$\{k \mid VM_k \in Pth\ \ cloud\ provider, P \in \{1,2,\dots,cp\}\}$= The set of VMs belonging to the pth provider
$Pcost_p$	The cost of one CPU for the *p*th provider (AUD/hour)
x_{ik}	1 if task *i* is assigned to VM_k and 0, otherwise

VM_{set}^{l} Set of underloaded VMs that are compatible with VM_l

5.1. The MOTS-PSO Objective Functions

In this study, the available resources in VMs that are determined to be possible destinations for the extra workload of a primary VM will be fewer than their allocated resources. Based on this fact, the objective functions of MOTS-PSO are formulated and explained below.

5.1.1. Task Transfer Time

When a task is assigned to a VM for execution, the input data of the task and the output data of its prerequisite tasks are uploaded to the VM from the corresponding storage node to the VM. Therefore, to transfer a task from one VM to another, the task and the output data produced by its prerequisite tasks should be transferred from the primary VM (VM_l) to the destination VM (VM_k) for execution. In this case, the total task transfer time for both the computing and data intensive tasks is estimated as follows, where the coefficient 1/8 is used to convert Megabit (Mb) to Megabyte (MB) where 1 Mb equals 1/8 MB:

$$Ttrans = \sum_{k=1}^{m}\sum_{i=1}^{n} \frac{x_{ik} * (DF_i + DI_i + \sum_{j \in PR_i} DO_j)}{\min(VM_{bw}^{l}, VM_{bw}^{k}) * (\frac{1}{8})} \quad (4)$$

5.1.2. Task Execution Cost and Time

The task execution cost (AUD per hour) for provider *p* is calculated as follows:

$$Cexe_p = \sum_{k \in SP_p} Pcost_p * Texe_k \quad (5)$$

where $Pcost_p$ is the cost of one CPU for the *p*th provider in AUD per hour, and $Texe_k$ is the estimated execution time (in hours) of the tasks assigned to each VM_k belonging to provider *p*.

The execution time of task *i* on VM_k can be estimated by dividing the total amount of CPU usage of task *i* (tcu_i) by the CPU speed of the corresponding VM ($VM_{CPUSpeed}^{k}$) i.e:

$$texe_k^i = \frac{tcu_i}{VM_{CPUSpeed}^{k}} \quad (6)$$

However, the CPU speed of the VM is not consistent during the task execution time in a VMware-vSphere based cloud environment with VMware-ESXi hypervisor. In this environment, the CPU speed changes based on the number of active CPUs because the VMware-ESXi divides the CPU speed of VM_k

($VM^k_{CPUSpeed}$) by the number of its active CPUs. To determine the execution time of a set of tasks scheduled to VM_k, therefore, the number of active CPUs during the execution time of these tasks should be estimated.

To estimate this number, we first determine the total number of tasks that are scheduled to VM_k as:

$$No^k_t = \sum_{i=1}^{n} x_{ik} \tag{7}$$

The integer division of No^k_t divided by VM^k_{ac} is calculated as follows, where VM^k_{ac} is the number of available CPUs allocated to VM_k:

$$Div^k_t = No^k_t \setminus VM^k_{ac} \tag{8}$$

This value is used to estimate how many tasks (i.e. $Div^k_t * VM^k_{ac}$) will be executed while all the CPUs of VM_k are active and the CPUs' speed is equal to $\frac{VM^k_{CPUSpeed}}{VM^k_c}$.

The remainder after the division of No^k_t by VM^k_{ac} (i.e. modulo) is calculated in Equation 9 to estimate how many CPUs will be used to execute the rest of the tasks. In this situation, the CPU speed of VM_k is divided by the number of its activate CPUs as calculated in Equation 10.

$$Mod^k_t = No^k_t \,\%\, VM^k_{ac} \tag{9}$$

$$N^k_{aCPU} = Mod^k_t + CN^k_{aCPU} \tag{10}$$

where CN^k_{aCPU} is the number of CPUs in VM_k that were busy before tasks were scheduled to it and is estimated as:

$$CN^k_{aCPU} = VM^k_c - VM^k_{ac} \tag{11}$$

For example, when we have 13 tasks scheduled to VM_k with four CPUs, three of which are available, then $Div_t^k = 13 \setminus 3 = 4$. This means that 4*3=12 tasks will be executed while all the CPUs of VM_k are busy, and the CPU speed is $\frac{VM_{CPUSpeed}^k}{4}$. In this example, $Mod_t^k = 13 \,\%\, 3 = 1$. This means that one of these tasks will be executed while the number of busy CPUs in VM_k is $Mod_t^k + CN_{aCPU}^k = 1 + (4 - 3) = 2$, and CPU speed equals $\frac{VM_{CPUSpeed}^k}{2}$ (see Figure 7).

$Texe_k$ is then estimated in hours by applying the aforementioned assumptions as follows:

$$Texe_k = \left(\frac{\sum_{i=1}^{Div_t^k * VM_{ac}^k} tcu_i}{\left(\frac{VM_{CPUSpeed}^k}{VM_c^k}\right)} + \frac{\sum_{i=(Div_t^k * VM_{ac}^k)+1}^{No_t^k} tcu_i}{\left(\frac{VM_{CPUSpeed}^k}{Mod_t^k + CN_{aCPU}^k}\right)}\right) * \frac{\rho}{60} \quad (12)$$

The value of $\rho = 0.33$ has been empirically determined.

We assume the same price for all CPUs in a VM, therefore task execution sequence and scheduling schema in each VM are not considered in this function. For instance, in a VM with three CPUs, the cost of assigning three tasks to three different CPUs will be the same as the cost of assigning the execution of three tasks to one CPU (see Equation 5). The total task execution cost for all providers is determined as:

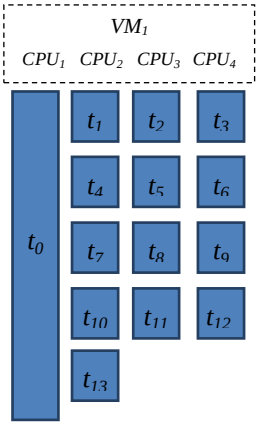


Figure 7: Task scheduling pattern

$$Cexe = \sum_{p=1}^{cp} Cexe_p \quad (13)$$

In situations where minimizing the execution time of tasks is more important than cost, the task execution time must be estimated. To do this, the value of coefficient $Pcost_p$ in Equation 5 for all providers is considered equal to 1; Equation 13 is then an estimation for the total task execution time in hours.

5.1.3. Power Consumption

Several power-aware multi-objective task scheduling models have been proposed for multi-core processors, grid and cloud environments [33, 34]. A variety of linear and non-linear objective functions have been suggested in these models to estimate power consumption based on task scheduling patterns, by considering the fact that energy will be reduced when the PM is either off or in idle mode [32, 35, 36]. It has also been proved by Buyya et al. [35, 37] that an idle server consumes around 70% of the power consumed by a fully utilized server. Previous studies show that having fewer active CPUs and PMs leads to lower power consumption in a cluster. Considering this fact, the ratio of active PMs and CPUs to all available PMs and CPUs has been minimized in this study to reduce power consumption. By applying this as an objective function, the optimization model avoids the selection of VMs on idle PMs as the destination for scheduled tasks and consequently reduces the power consumed in the corresponding cloud cluster. To calculate this ratio, we first determine the power consumption for each fully utilized PM (i.e. all of its CPUs are active) as follows, based on the fact that the power consumed by the PM is linear with the number of busy/active CPUs [38] and increases by this number:

$$Pw_{ful}(PM_z) = Pw_0^z + \alpha\ N_{CPU}^z \tag{14}$$

where Pw_0^z is the amount of power that PM_z consumes when all of its CPUs are idle, N_{CPU}^z is the number of CPUs of fully utilized PM_z that are active, and α is the amount of extra power consumed above Pw_0 for every active CPU of PM_z. Using this, the value of α is:

$$\alpha = \frac{Pw_{ful}(PM_z) - Pw_0^z}{N_{CPU}^z} \tag{15}$$

For a cluster with homogenous PMs, the value of α and Pw_0 are the same for every PM. Therefore, the total amount of consumed power in this cluster can be determined as follows:

$$Pw_{ful}(Cluster) = \sum_{z=1}^{N_{PM}} Pw_{ful}(PM_z) = \sum_{z=1}^{N_{PM}} Pw_0 + \alpha \sum_{z=1}^{N_{PM}} N_{CPU}^z \tag{16}$$

then

$$Pw_{ful}(Cluster) = Pw_0 * N_{PM} + \alpha * N_{CPU} \tag{17}$$

where N_{CPU} is the total number of CPUs in the cluster and can be calculated by counting the number of CPUs allocated to each VM i.e. $N_{CPU} = \sum_{k=1}^{m} VM_c^k$. Using this $Pw_{ful}(Cluster)$ is calculated as:

$$Pw_{ful}(Cluster) = Pw_0 * N_{PM} + \alpha * \sum_{k=1}^{m} VM_c^k \quad (18)$$

By applying the same logic, the value of the power consumed by the active PMs and CPUs in this cluster is calculated as follows:

$$Pw_{active}(Cluster) = Pw_0 * N_{aPM} + \alpha * \sum_{k=1}^{m} N_{aCPU}^k \quad (19)$$

where N_{aCPU}^k is the number of active CPUs in VM_k. To calculate N_{aCPU}^k, the number of CPUs of VM_k that are already busy is calculated by applying Equation 11, and the number of activated CPUs is considered. The number of tasks assigned to VM_k may be less than the number of available CPUs in VM_k, or it may exceed this number, therefore the number of activated CPUs in VM_k after a task has been scheduled is equal to the minimum value of the total number of tasks assigned to VM_k and the total number of available CPUs in this VM. The total number of active CPUs in VM_k can be estimated as:

$$N_{aCPU}^k = CN_{aCPU}^k + min\left(\sum_{i=1}^{n} x_{ik}, VM_{ac}^k\right) \quad (20)$$

N_{aPM} is the number of active PMs in all iterations and is estimated on the basis that PM_z will be activated if at least one of its allocated VMs is active. The activation status of VMs is determined according to the current number of busy CPUs (CN_{aCPU}^k) and the number of tasks assigned to them. VM_k is already activated if at least one of its CPUs is busy before a task is scheduled to it. The current status of VM_k is then calculated as:

$$CVM_{status}^k = \min(CN_{aCPU}^k, 1) \quad (21)$$

In the situation where all CPUs of VM_k are idle (available), VM_k will be activated by having at least one task assigned to it, and the status of VM_k is determined by the following formula:

$$VM_{status}^k = \min\left(\sum_{i=1}^{n} x_{ik}, 1\right) \quad (22)$$

where $\sum_{i=1}^{n} x_{ik}$ is the number of tasks assigned to the available CPUs of VM_k. By applying Equations 21 and 22, the number of active VMs located on PM_z is calculated as follows:

$$N_{aVM}^z = \sum_{k \in Svm_z} \left(CVM_{status}^k + (1 - CVM_{status}^k) * VM_{status}^k\right) \quad (23)$$

and the number of active PMs is determined using the following formula:

$$N_{aPM} = \sum_{z=1}^{Npm} \min(\, N_{aVM}^{z}, 1) \tag{24}$$

Using Equations 18 and 19, the ratio of consumed power in the cluster for each task scheduling pattern can be calculated as:

$$\frac{Pw_{active}(Cluster)}{Pw_{ful}(Cluster)} = \frac{Pw_0 * N_{aPM} + \alpha * \sum_{k=1}^{m} N_{aCPU}^{k}}{Pw_0 * N_{PM} + \alpha * \sum_{k=1}^{m} VM_c^k} \tag{25}$$

$$= \frac{Pw_0 \,(N_{aPM} + \frac{\alpha}{Pw_0} * \sum_{k=1}^{m} N_{aCPU}^{k})}{Pw_0 (N_{PM} + \frac{\alpha}{Pw_0} * \sum_{k=1}^{m} VM_c^k)}$$

As a result, the following formula is developed as an objective function in our task scheduling model to control power consumption in the cluster:

$$PowerC = \frac{(N_{aPM} + \mu \sum_{k=1}^{m} N_{aCPU}^{k})}{(N_{PM} + \mu \sum_{k=1}^{m} VM_c^k)} \tag{26}$$

where $\mu = \frac{\alpha}{Pw_0^z}$. For instance, for one of our servers in this study (Altix XE320 [39]) with $Pw_{ful}(PM) = 115kw$, $Pw_0 = 40kw$ [40], and 16 CPUs, the value of μ is equal to 0.1.

5.1.4. Length of VM Task Queue

For each possible best solution of the multi-objective task scheduling pattern, extra tasks will be allocated to the task queue of VM_k if the number of tasks assigned to VM_k exceeds the number of its CPUs (see Figure 8). In this model, we consider another objective function to optimize the task scheduling pattern by minimizing the length of the VM task queues. This will reduce the makespan of the corresponding jobs to the tasks, and consequently reduce response time, as fewer tasks will be located in queues and in waiting mode.

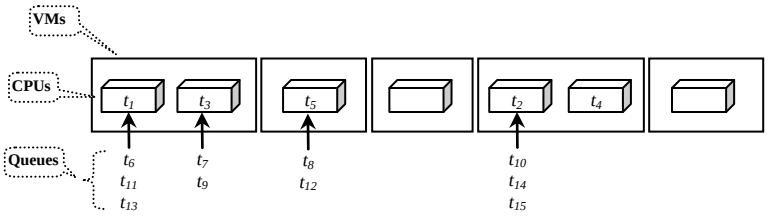


Figure 8: A sample task scheduling pattern between VMs

The following objective function is proposed to minimize the number of tasks in queues that will be created in any possible optimal task scheduling pattern:

$$\gamma_k = \sum_{i=1}^{n} (x_{ik} * \frac{1}{(RVM_m^k)_i * (RVM_c^k)_i}) \quad (27)$$

where i is the task index, and the value of $(RVM_m^k)_i$ and $(RVM_c^k)_i$ is the portion of remaining memory and CPU capacity of VM_k respectively after VM_k has received each scheduled task, calculated as:

$$(RVM_m^k)_i = \frac{VM_{am}^k - \left(\sum_{j=1}^{i-1} x_{jk} * tm_j\right)}{VM_{am}^k} \quad (28)$$

$$(RVM_c^k)_i = \frac{VM_{ac}^k - \left(\sum_{j=1}^{i-1} x_{jk} * tc_j\right)}{VM_{ac}^k} \quad (29)$$

where the values of $\left(\sum_{j=1}^{i-1} x_{jk} * tm_j\right)$ and $\left(\sum_{j=1}^{i-1} x_{jk} * tc_j\right)$ are the total amount of memory and number of CPUs required to execute the tasks previously assigned to VM_k prior to task i.

To calculate γ_k, the formula proposed in [41-43] is improved by considering changes in the capacity of available VMs after the assignation of new tasks. Equation 27 indicates that scheduling task i to VM_k —which reduces the amount of available CPU and memory of the corresponding VM— negatively affects VM_k's performance and the execution time of its tasks. In this formula, if $VM_{am}^k \leq \left(\sum_{j=1}^{i-1} x_{jk} * tm_j\right)$ and/or $VM_{ac}^k \leq \left(\sum_{j=1}^{i-1} x_{jk} * tc_j\right)$, the implication is that there is no available memory or CPU to execute extra tasks, thus the suggested solution will increase the number of accumulated tasks in the task queue of VM_k and create a long task queue for this VM. This study prevents the suggested solution from being chosen as an optimal task scheduling solution by assigning a big penalty value to the value of γ_k. Using this, γ_k will not be minimized and the probability of choosing the corresponding task scheduling solution as an optimal solution will be reduced. To increase the value of γ_k, a small value is assigned to $(RVM_m^k)_i$ and/or $(RVM_c^k)_i$. This is achieved as follows: if $(RVM_m^k)_i$ and/or $(RVM_c^k)_i$ are equal to zero, the value of 10^{-3} will be added to them, and if each of them has a negative value, they will be converted to $(RVM_m^k)_i^{-1}$ and/or $(RVM_c^k)_i^{-1}$ then multiplied to a small value (-10^{-3}). This method for panelizing solutions with negative values for $(RVM_m^k)_i$ and/or $(RVM_c^k)_i$ has been determined to not only penalize solutions that do not achieve optimal resource utilization and increase the task queue length of the corresponding VMs, but also to enable the system to rank these suggested solutions. For example, we assume in a solution that $(RVM_c^k)_i = -2$ (i.e. we have two more CPUs required to execute task i), and in another solution $(RVM_c^k)_i = -3$. Although the task scheduling pattern suggested in both solutions will increase the task queue length of the corresponding VM, the first solution is better than the second solution as it requires fewer extra CPUs. Therefore, we change the value of $(RVM_c^k)_i$ in first solution to $(RVM_c^k)_i =$

$-10^{-3}/_{-2}$, and in the second solution to $(RVM_c^k)_i = -10^{-3}/_{-3}$. Considering this fact that ($10^{-3}/_2 > 10^{-3}/_3$), the value of γ_k in the first solution is less than the value obtained in the second solution. Therefore, the first solution has a higher rank than the second solution in terms of minimizing γ_k.

Since the values of $(RVM_m^k)_i$ and $(RVM_c^k)_i$ are multiplied by each other, using this will increase the value of γ_k dramatically whenever one of them equals zero or has a negative value. Therefore, the probability of this pattern being chosen as a possible optimal solution will decrease. This prevents the assignation of tasks to VMs that do not have available resources. The following formula is then used as an optimization objective function in our proposed model to minimize VM task queue length and optimize the load balance:

$$\Gamma = \sum_{k=1}^{m} \gamma_k \tag{30}$$

5.1.5. The Multi-Objective Problem

The multi-objective optimization problem for task scheduling applied by the MO-LB system is defined as follows, based on the determined objective functions:

Problem:

$$min\ f_{TransferTime} = Ttrans \tag{31}$$

$$min\ f_{Excecution} = Cexe \tag{32}$$

$$min\ f_{PowerCunsumption} = PowerC \tag{33}$$

$$min\ f_{TaskQueue} = \Gamma \tag{34}$$

Subject to

$$\sum_{k=1}^{m} x_{ik} = 1, \forall\ i = 1, \dots, n$$

$$x_{ik} \in \{0,1\}, \forall i = 1, .., n\ \&\ k = 1, \dots, m$$

$$N_{aPM} \leq N_{PM}$$

$$N_{aCPU}^k \leq VM_{ac}^k, k = 1, \dots, m$$

5.2. The MOTS-PSO Algorithm

The task scheduling optimization model is an NP-Complete problem in cloud computing, and the supremacy of PSO for such optimization problems in the cloud and grid environment has been proved in [41, 42, 44, 45]. In addition, the efficiency of MOPSO for solving multi-objective task scheduling problems has been examined in [29] in comparison with the multi-objective genetic algorithm. It has been shown in [29] that MOPSO is the most efficient and reliable algorithm for solving these problems because it not only determines the optimal task scheduling pattern with the highest QoS, but also obtains the solution in the shortest possible time. In light of these results, MOPSO is applied in this study to find the optimal schema for transferring tasks from primary VMs to the selected destination VMs. For small numbers (e.g., 20 Tasks and 10 VMs) the convergence time of MOTS-PSO is less than 0.3 seconds. For industry sized deployments (e.g., 100 Tasks and 20 VMs), and for very large numbers of tasks and VMs (e.g., 1000 Tasks and 50VMs), the computation can be parallelized by processing MOTS-PSO in parallel for smaller groups of tasks and VMs. In this situation, the tasks and VMs are categorized in several sets with a small number of tasks and VMs. MOTS-PSO is then run in parallel for each set and the time consumed for task scheduling does not increase.

In this section, the preliminary definition of the PSO method is first described. The MOTS-PSO algorithm is then explained to complete Step 6 of the MO-LB algorithm and solve the formulated task scheduling optimization problem.

5.2.1. Particle Swarm Optimization

In the majority of optimization problems, the objective functions are in conflict with each other and there is no unique solution for them. Therefore, the goal is to find good trade-off solutions that represent the best possible compromises among the objectives [46]. A multi-objective optimization problem is defined as follows:

$$\text{Min} \quad \vec{F}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), \dots, f_k(\vec{x})] \tag{35}$$

where $\vec{X} = (x_1, x_2, \dots, x_k)$ is the vector of decision variables; $f_i: R^n \to R, i = 1, \dots, k$ are the objective functions. Let particle $\vec{X}_1 = (x_1, x_2, \dots, x_k)$ represent a solution to (1). A solution $\vec{X}_2$ dominates $\vec{X}_1$ if $f_j(\vec{X}_1) \geq f_j(\vec{X}_2)$ for all j=1,..,k and $f_j(\vec{X}_1) > f_j(\vec{X}_2)$ for at least one j=1,…,k. A feasible solution $\vec{X}_1$ is called Pareto optimal (non-dominated) if there is no other feasible solution $\vec{X}_2$ that dominates it. The set of all objective vectors $F(\vec{X}_1)$ corresponding to the Pareto optimal solutions is called the Pareto front

(P*). Thus, the aim is to determine the Pareto optimal set from the set F of all the decision variable vectors (particles) [47-51].

Particle Swarm Optimization (PSO) is a population-based search algorithm based on a simulation of the social behavior of birds which was originally proposed by Kennedy and Eberhart [52]. Although originally adopted for balancing weights in neural networks, PSO soon became a very popular global optimizer, mainly in problems in which the decision variables are real numbers [53, 54]. In PSO, particles are flown through hyper-dimensional search space. Changes to the position of the particles within the search space are based on the socio-psychological tendency of individuals to emulate the success of other individuals. The position of each particle is changed according to its own experience and that of its neighbors. Let $\vec{\mathrm{X}}_{\mathrm{i}}(\mathrm{t})$ denote the position of particle *i*, at iteration *t*. The position of $\vec{\mathrm{X}}_{\mathrm{i}}(\mathrm{t})$ is changed by adding a velocity $\vec{\mathrm{V}}_{\mathrm{i}}(\mathrm{t}+1)$ to it, i.e.:

$$\vec{\mathrm{X}}_{\mathrm{i}}(\mathrm{t}+1) = \vec{\mathrm{X}}_{\mathrm{i}}(\mathrm{t}) + \vec{\mathrm{V}}_{\mathrm{i}}(\mathrm{t}+1) \tag{36}$$

The velocity vector reflects the socially exchanged information and, in general, is defined in the following way:

$$\vec{\mathrm{V}}_{\mathrm{i}}(\mathrm{t}+1) = \mathrm{W}\vec{\mathrm{V}}_{\mathrm{i}}(\mathrm{t}) + \mathrm{C}_1\mathrm{r}_1\left(\vec{\mathrm{x}}_{\mathrm{pbest}_{\mathrm{i}}} - \vec{\mathrm{X}}_{\mathrm{i}}(\mathrm{t})\right) + \mathrm{C}_2\mathrm{r}_2\left(\vec{\mathrm{x}}_{\mathrm{gbest}_{\mathrm{i}}} - \vec{\mathrm{X}}_{\mathrm{i}}(\mathrm{t})\right) \tag{37}$$

where C_1 is the cognitive learning factor and represents the attraction a particle has to its own success; C_2 is the social learning factor and represents the attraction a particle has to the success of the entire swarm; *W* is the inertia weight, which is employed to control the impact of the previous history of velocities on the current velocity of a given particle; $\vec{\mathrm{x}}_{\mathrm{pbest}_{\mathrm{i}}}$ is the personal best position of the particle *i*; $\vec{\mathrm{x}}_{\mathrm{gbest}}$ is the position of the best particle of the entire swarm; and $\mathrm{r}_1, \mathrm{r}_2 \in [0,1]$ are random values [46, 49, 50] .

5.2.2. The MOTS-PSO Algorithm

The MOTS-PSO algorithm specifies the most appropriate VMs to which the tasks of the primary VMs can be allocated, and finds the optimal task scheduling schema by applying the PSO algorithm adopted from [55] and modified to MOPSO to support multi-objective problems. This algorithm applies the data and information determined in Table 2, and the output variables obtained by Steps 1, 2, 4 and 5 of the MO-LB algorithm as its input variables.

In this stage, CUP output variables are generated as input variables for MOTS-PSO: a set of primary VMs, and a set of destination VMs as $VM_{set} = \{vm_1, \dots, vm_l\}$ which have available memory and CPUs for executing extra tasks. For each primary VM (VM_l) in the MOTS-PSO algorithm, a set of compatible

VMs from VM_{set} are chosen as new destinations for the extra workload (VM^{l}_{set}). The set of tasks to be transferred from VM_l are determined as $T^{l}_{set} = \{t_1, \dots, t_n\}$. Lastly, the MOPSO algorithm is applied to find the best solution for the multi-objective task scheduling optimization problem (see Section 5.1) to schedule tasks in T^{l}_{set} to VMs in VM^{l}_{set}. All particle positions $\vec{X}_i = (x_1, x_2, \dots, x_n)$ determined by MOPSO by applying Equations 36 and 37 are vectors with continuous values, but their corresponding discrete values are needed to determine the index of VMs chosen for task execution. The Smallest Position Value (SPV) rule is proposed in [56] to modify the PSO algorithm and enable the continuous PSO algorithm to be applied to all classes of sequencing problems such as the task scheduling problem, which are NP-hard. By using the SPV rule, permutation can be determined through the position values of the particle so that the positions of each particle are updated at each iteration *k* in the PSO algorithm and the fitness value of the particle can be computed with that permutation. The feasibility of PSO using the SPV rule to determine the optimal solution is proved in [56]. In addition, the SPV rule is widely applied to convert the vector of the particles' continuous position vector to discrete vectors [41, 56-59]. Given this knowledge, the SPV rule is applied in this study. First a new sequence vector $S(\vec{X}_i) = (s_1, s_2, \dots, s_n)$ is generated by applying the SPV rule based on continuous vector $\vec{X}_i = (x_1, x_2, \dots, x_n)$. For example, the smallest position value for continuous vector $\vec{X}_i = (1.8, -0.99, 3.01, -0.7, -1.2, 2.15)$ is -1 2 0. Therefore, the dimension 5 is assigned as s_1. The second smallest position value is -0.99, so s_2= 2, and so on. For a task scheduling problem with *n* tasks and *m* VMs, discrete vectors $d(\vec{X}_i) = (d_1, d_2, \dots, d_n)$ is generated by applying the following equation [57]:

$$d_i = s_i \bmod m \tag{38}$$

For example, for continuous vector $\vec{X}_i = (1.8, -0.99, 3.01, -0.7, -1.2, 2.15)$ in a task scheduling problem with six tasks and three VMs, $\vec{S}_i = (5, 2, 4, 1, 6, 3)$ and $\vec{d}_i = (2, 2, 1, 1, 0, 0)$. Based on this solution ($\vec{d}_i$), tasks 1, 2, 3, 4, 5 and 6 are assigned to VMs with indices 2, 2, 1, 1, 0 and 0 respectively.

Ultimately, in MOTS-PSO algorithm, the current value of VM properties (CPU, memory, etc.) will be updated on the GB. In summary, the following steps should be conducted for each primary VM by the MOTS-PSO algorithm to complete Step 6 of the MO-LB system:

Algorithm3. The MOTS-PSO algorithm (Step 6 of the MO-LB algorithm)

Input: All variables in Table 2, and variables created in Steps 1, 2, 4 and 5 of the MO-LB algorithm.

[Begin MOTS-PSO algorithm]

7.1. Choose a VM from a set of primary VMs suggested by the CUP algorithm as VM_l.

7.2. Choose a set of compatible VMs as $VM_{set}^{l} = \{vm_1, \dots, vm_m\}$ form the new candidate destination VMs ($VM_{set} = \{vm_1, \dots, vm_l\}$) determined by the CUP algorithm.

7.3. Determine the set of tasks to be transferred from the primary VMs as $T_{set}^{l} = \{t_1, \dots, t_n\}$

7.4. Apply the PSO method to find the Pareto optimal schema to assign the determined tasks (T_{set}^{l}) to the specified VMs (VM_{set}^{l}) minimizing task transfer time, task execution cost/time, length of VM task queue, and power consumption as follows:

7.4.1. Create an initial population array of every particle *i* ($\vec{X}_i$) with random positions and velocities on *n* dimensions in the search space.

7.4.2. Initialize an archive in which members are non-dominated solutions (n dimensions particles/genes whose position/pattern is a Pareto optimal solution)

7.4.3. Determine the value of $DO_i, DI_i, DF_i, VM_m^k, VM_c^k$ and VM_{bw}^k based on $d(\vec{X}_i)$ to calculate the value of every fitness function.

7.4.4. For each particle, calculate fitness functions $f_{TransferTime}$, $f_{Execution}$, $f_{TaskQueue}$ and $f_{PowerConsumption}$ by applying Equations 31, 32, 33 and 34.

7.4.5. For each particle, evaluate the desired optimization fitness functions.

7.4.6. Update the archive content by deleting dominated members from the archive and storing the Pareto optimal (non-dominated) solutions in the archive.

7.4.7. Sort archive members based on the number of optimized objective functions and their determined weight.

7.4.8. Compare each particle's fitness evaluation with its personal best fitness function value ($\vec{x}_{pbest_i}$). If the current value is better than $\vec{x}_{pbest_i}$, then set $\vec{x}_{pbest_i}$equal to the current value, and the best position *pi* equal to the current location $\vec{x}_i$ in *n*-dimensional space.

7.4.9. Choose $\vec{X}_{gbest}$ from top sorted members in the archive as the best global position.

7.4.10. Change the velocity and position of the particle according to Equations 36 and 37.

7.4.11. Convert continuous position values vector of $\vec{X}_i$ to discrete vector $d(\vec{X}_i)$ using SPV rule to determine the allocated VM for every arrival task.

7.4.12. If a criterion is met (usually a sufficiently good fitness or a maximum number of iterations) then

7.4.12.1. Output the best particle position in *n*-dimensional space $d(\vec{X}_{gbest})$ as the optimal task migration schema

Else

7.4.12.2. Go to Step 7.4.3

7.5. Calculate and update the current VM properties according to the optimal task scheduling solution.

7.6. Transfer tasks and their corresponding data to the destination VMs

[End MOTS-PSO algorithm]

6. Evaluation Results

We compare the efficiency of the MO-LB system with the VMware-ESXi auto load balancing system which is based on VM migration, by implementing those systems in a VMware-vSphere based private cloud. HTCondor [25] is also applied to implement the suggested solution and submit tasks to the destination VMs. HTCondor is a software system that creates a High-Throughput Computing (HTC) environment. When a user submits a job to HTCondor, HTCondor finds an available machine on the network and begins to run the job on that machine. It can checkpoint the job and migrate jobs to a different machine. HTCondor implements ClassAds, a clean design that simplifies the user's submission of jobs. ClassAds work in a fashion similar to 'want ads' in classified advertising. All machines in the HTCondor pool advertise their resource properties, both static and dynamic, such as available RAM memory, CPU type, CPU speed, virtual memory size, physical location, and current load average, in a resource offer advertisement (ad). A user specifies a resource request ad when submitting a job. The request defines both the required and desired properties of the resource for running the job. HTCondor acts as a broker by matching and ranking resource offer ads with resource request ads, making certain that all requirements in both ads are satisfied. During this match-making process, HTCondor also considers several layers of priority values: the priority the user has assigned to the resource request ad, the priority of the user who submitted the ad, and the preference of the machines in the pool to accept certain types of ads over others [60]. In this study, the destination machines are determined on the basis of the optimal solution suggested by the MOTS-PSO sub-system. To achieve this, the names of the selected machines are detailed in the submit file of their corresponding jobs, and HTCondor sends the jobs to the specified machines.

6.1. Environment Description

The cloud environment is designed to have two data-stores, four PMs, twenty VMs, two cloud providers and 200 arrival computation, memory and data intensive tasks that are independent. We ered $Pcost_1 = Pcost_2 = 1$ in Equation 5, therefore the objective function $f_{Excecution}$ (Equation 32) is applied as an estimation of the task execution time. The information about VMs and tasks is summarized in Tables 3 and 4. To evaluate the proposed model, several computational, memory and data intensive tasks are generated, using C++ programming language, as examples of the different types of task that are executed in a BoT as part of the workflow applications, such as Pegasus workflow applications. In this study,

the computationally intensive tasks multiply large matrices. The memory intensive tasks consume memory by inserting and deleting data to and from memory blocks. The data intensive tasks transfer large images between primary and destination VMs. The PMs are homogenous and each has a different type of VM (see Table 5).

Table 3. Properties of VMs

VM Id	CPU speed in GHz ($VM_{CPUSpeed}$)	Available memory in MB (VM_m)	Bandwidth in Mb/s (VM_{bw})	Number of CPUs (VM_c)	OS
1-4	2.6	4096	1024	4	Ubuntu Linux
5-8	2.6	4096	1024	2	Ubuntu Linux
9-12	1.3	2048	1024	2	Ubuntu Linux
13-16	1.3	1024	1024	1	Ubuntu Linux
17-20	1.3	512	1024	1	Ubuntu Linux

Table 4. Properties of tasks

Task Id	File Size in kB (*DF*)	Output Size in Byte (*DO*)	Input size in MB (*DI*)	Required CPUs (t_c)	CPU usage in GHz (t_{cu})	Total memory usage in MB	Max level of memory usage in MB (t_m)
Computationally Intensive Tasks							
1-20	8.7	47	0	1	186.372	2055.06	125.828
21-40	8.5	46	0	1	21.186	985.616	62.912
41-60	8.5	47	0	1	67.166	754.924	62.912
61-80	8.5	46	0	1	8.261	471.848	73.4
81-100	8.5	47	0	1	21.759	524.26	62.912
101-120	8.5	46	0	1	2.263	125.82	41.94
Memory Intensive Tasks							
121-140	9.2	170	0	1	41.097	11534.304	943.716
141-160	9.2	169	0	1	38.367	9940.468	859.832
161-180	9.2	170	0	1	38.372	1992.268	167.772
Data Intensive Tasks							
181-200	8.5	46	1.4	1	2.833	377.484	125.828

Table 5. Resource allocation in the cluster

PM Id	VMs				
1	VM_1	VM_5	–	–	–

2	VM_2	VM_6	VM_{10}	VM_{14}	VM_{18}
3	VM_3	VM_7	VM_{11}	VM_{15}	VM_{19}
4	VM_4	VM_8	VM_{12}	VM_{16}	VM_{20}

6.2. Scenario development

In this study, we evaluate the MO-LB system in cases when VM migration is required, i.e. (1) a host PM is likely to be overloaded (PM hotspot), or (2) a VM needs to be scaled up. The hotspot situation occurs when all the resources in the host PM are allocated to VMs, and all VMs are using their maximum capacity to execute their allocated tasks. In this case —irrespective of whether the VMs are used for SaaS, PaaS or IaaS— the PM is at risk of being overloaded. The current solution for this problem, conducted by the Auto Load Balancer (ALB) of VMware-ESXi, is to migrate a set of VMs from the overloaded PM to other PMs that have more available resources [3-7, 14-18]. The second situation investigated in this study occurs when VMs that are delivered as IaaS exhibit low performance and their client asks for VM scale-up when no capacity is available in their host PM. In this case, VMware-ESXi migrates the VMs to PMs that have the capacity and scales up these VMs [3-7, 14-18].

In contrast, the MO-LB solution for the first situation is to select a set of VMs that deliver SaaS or PaaS and are located on the overloaded PM, stop these VMs from working, and transfer their tasks in progress and all the tasks that have accumulated in their task queues to compatible VMs with available capacity located on other PMs. This reduces resource utilization on the overloaded PM and eases the tension between its allocated VMs. In the second situation —where a VM that delivers IaaS is performing poorly and needs to be scaled up— MO-LB applies the same process. The resources allocated to the selected VMs are released and allocated to the poorly performing VM, which is thus scaled up.

The ALB of VMware-ESXi reacts after a number of VMs start to exhibit low performance and the PM utilizes more than the determined threshold of its capacity. However, the prediction model applied in the MO-LB system predicts VM performance and reacts before the corresponding PM meets the predefined utilization threshold.

In this study, the MO-LB system is compared to VMware-ESXi in both live and storage migration. In live migration, the execution state and active memory of a VM is migrated to a new ESX host. Live storage VM migration involves changing both the host and the data store. In this case, the disk files of the VM are also migrated.

6.3. Implementation

To implement the MO-LB system, we first randomly schedule the determined tasks to the VMs in our private cloud by applying HTCondor functionality for one week to create historical data of CPU usage (VM^{k}_{Ucpu}) as the historical input data for the CUP sub-system. The CUP algorithm (Algorithm 2) is implemented using MATLAB.

6.3.1. Implementing the MO-LB System

To evaluate the MO-LB system, we apply a PM with the lowest capacity compared to other PMs to give us the ability to overload this PM. To emulate a situation in which a PM will be overloaded, random tasks are chosen from the list of computationally intensive tasks in Table 4 and scheduled to the VMs located on PM_1, as illustrated in Table 6.

Table 6. List of tasks scheduled to VMs located on PM_1

VM ID	Tasks										
1	t_5	t_{11}	t_{41}	t_{45}	t_{47}	t_{49}	t_{30}	t_{21}	t_{25}	t_{27}	t_{50}
5	t_{44}	t_{46}	t_{51}	t_{27}	t_{39}	–	–	–	–	–	–

In addition, a random set of tasks is scheduled to other VMs in the cluster. The CUP algorithm also checks the performance of every VM every two minutes (as previously described). After two minutes, the CUP algorithm specifies a list of possible overloaded and underloaded VMs in the cluster as follows:

Table 7. CUP results

VM ID	$fCA'^{k}_{cpu}(x)$		$VM^{k}_{cpu}(ct)$		$\boldsymbol{VM^{k}_{load}}$	**Number of available CPUs (VM^{k}_{ac})**
1	≥ 0	and	≥ 80	then	Overloaded	0
5	≥ 0	and	≥ 80	then	Overloaded	0
3	≤ 0	and	≤ 80	then	Under-loaded	2
6	≤ 0	and	≤ 80	then	Under-loaded	2
7	≤ 0	and	≤ 80	then	Under-loaded	2
9	≤ 0	and	≤ 80	then	Under-loaded	1
10	≤ 0	and	≤ 80	then	Under-loaded	1
11	≤ 0	and	≤ 80	then	Under-loaded	1

As can be seen from Table 7, CUP suggests that VM_1 and VM_5 , located on PM_1, are poorly performing VMs, which means that PM_1 is at risk of becoming overloaded. VM_3, VM_9 and VM_{11} from the list of high performance VMs are compatible with VM_1 and have 2, 1 and 1 available CPUs respectively. VM_6, VM_7 and VM_{10} are compatible with VM_5 with 2, 2 and 1 available CPUs respectively. In this situation, the VM with the lowest number of scheduled tasks is selected to cease executing, and all the tasks scheduled to this VM (pending and executing tasks) are transferred to reduce the tension between VMs on the overloaded PM. The executing tasks on this VM will resume and continue their execution on the destination VM from where they left off, using the HTCondor check point mechanism. The VMs that have lower numbers of scheduled tasks are chosen to cease executing so that fewer tasks need to be transferred. The tasks that have accumulated in the VM_1task queue and all the tasks scheduled to VM_5 are determined as $T^1_{set} = \{t_{47}, t_{49}, t_{30}, t_{21}, t_{25}, t_{27}, t_{50}\}$ and $T^5_{set} = \{t_{44}, t_{46}, t_{51}, t_{27}, t_{39}\}$. The MOTS-PSO algorithm (Algorithm 3) is then used to schedule T^1_{set} and T^5_{set} to their compatible VMs. The scheduling results are illustrated in Tables 8 and 9.

Table 8. Optimal suggested pattern for scheduling T^1_{set} to VM^1_{set}

Tasks	t_{47}	t_{49}	t_{30}	t_{21}	t_{25}	t_{27}	t_{50}
Task CPU usage (GHz)	67.166	67.166	21.186	21.186	21.186	21.186	67.166
VMs	VM_{11}	VM_3	VM_9	VM_3	VM_3	VM_9	VM_3

Table 9. Optimal suggested pattern for scheduling T^5_{set} to VM^5_{set}

Tasks	t_{44}	t_{46}	t_{51}	t_{27}	t_{39}
Task CPU usage (GHz)	67.166	67.166	67.166	21.186	21.186
VMs	VM_6	VM_7	VM_7	VM_6	VM_{10}

The optimal values for the objective functions $f_{TransferTime}$, $f_{Excecution}$, $f_{TaskQueue}$ and $f_{PowerConsumption}$ of MOTS-PSO for the optimal pattern to schedule T^1_{set} to $VM^1_{set}$are 0 seconds, 2.42 hours (02:25:12), 2.58 and 1 respectively. The corresponding values to these objective functions determined for scheduling T^5_{set} to VM^5_{set} are 0 seconds, 1.12 hours (01:07:12), 7.20 and 1 respectively.

The execution of tasks t_{44} and t_{46} on VM_5 are stopped, and T^1_{set} and T^5_{set} are scheduled according to the suggested optimal patterns to their specified destinations. Following the execution of these tasks,

the related data about the evaluation measurements are determined and summarized in Table 10. Each set of tasks (e.g., T_{set}^{1}) is considered as a job, and each sub-set of this set of tasks that is scheduled to a different destination VM (i.e.$\{t_{49}, t_{50}, t_{21}, t_{25}\}$, $\{t_{27}, t_{30}\}$, and $\{t_{47}\}$) is considered as a sub-job. The value of job makespan for T_{set}^{1} is calculated as the maximum makespan among these sub-jobs on their corresponding VM (i.e. VM_3, VM_9, and VM_{11}). The job makespan for set q is calculated as:

$$Jmakespan_q = max_{k=1}^{m}[\, Makespan_k^{sub-job_p}] \tag{39}$$

where $Makespan_k^{sub-job_p}$ is the maximum time taken by the claimed CPUs of VM_k to execute sub-job p of job q, and is calculated as:

$$Makespan_k^{sub-job_p} = max_{i=1}^{VM_{ac}^k}\left(Texe_{CPU_i}^k\right) \tag{40}$$

where $Texe_{CPU_i}^k$ is the task execution time on *ith* available CPU on VM_k .The value of $Makespan_k^{sub-job_p}$ for each VM is determined by monitoring the VM performance and checking its task log files.

The job execution time for each set is the summation of the execution time of its sub-jobs on their destination VMs. It can be seen from the results that the estimated execution time calculated as objective function $f_{Excecution}$ for the tasks scheduled to VM_1 and VM_5 is close to their corresponding real time consumption of 02:19:06 and 01:00:23 respectively.

In total, the 07:39:29 hours taken for all jobs to be executed with 01:34:00 hours makespan which is the maximum value among $Jmakespan_1, Jmakespan_2, Jmakespan_3$ and $Jmakespan_4$ (see Table 10).

Table 10. MO-LB implementation results

VM ID	Scheduled Tasks	Sub-job Execution Time	Sub-job Makespan
	Executing tasks by VM_1		
1	$t_5, t_{11}, t_{41}, t_{45}$	04:16:00	01:34:00
		Job$_1$ execution time = **04:16:00**	Jmakespan$_1$= **01:34:00**
	Scheduling T_{set}^{1} to VM_{set}^{1}		
3	$t_{49}, t_{50}, t_{21}, t_{25}$	01:26:44	00:43:20
9	t_{27}, t_{30}	00:21:30	00:21:30
11	t_{47}	00:30:52	00:30:52
		Job$_2$ execution time = **02:19:06**	Jmakespan$_2$= **00:43:20**
	Executing tasks by VM_5 before transferring its tasks		

5	t_{44}, t_{46}	00:04:00	00:02:00
		Job_3 execution time = **00:04:00**	$Jmakespan_3$= **00:02:00**
	Scheduling T_{set}^5 to VM_{set}^5		
6	t_{44}, t_{27}	00:19:42	00:14:55
7	t_{46}, t_{51}	00:31:30	00:16:30
10	t_{39}	00:10:11	00:10:11
		Job_4 execution time = **01:00:23**	$Jmakespan_4$= **00:16:30**
Total		**07:39:29**	**01:34:00**

6.3.2. Implementing VMware Auto Load Balancer

After implementing the MO-LB model, the cluster is set to apply the VMware ALB. The tasks are scheduled to VMs in the same order as scheduled in Section 6.3.1, and VM_1 and VM_5 allocated on PM_1 start to use all the resource capacity of their host. After five minutes, when VM_1 has $\{t_{47}, t_{49}, t_{30}, t_{21}, t_{25}, t_{27}, t_{50}\}$ in its task queue and is executing t_5, t_{11}, t_{41} and t_{45}, and VM_5 has $\{t_{51}, t_{27}, t_{39}\}$ in its task queue and is executing t_{44} and t_{46}, VM_5 is migrated to PM_2 by the hypervisor in approximately three seconds. In this state, VM_5has approximately 100 Mb active memory which is migrated to the new host. In the same situation, the live storage migration of VM_5 with 290 GB (296960 Mb) disk file size and 100 Mb active memory takes approximately 5.5 minutes. Before VM_5 migration, the CPU speed of VM_1 and VM_5 were less than the guaranteed speed; therefore, it takes longer than expected for the executing tasks to be completed. In addition, VM_5 is slowed down during the storage migration time, which also affects the execution time of its allocated task.

The two sets of tasks that are scheduled to VM_1 and VM_5 are considered as two jobs. The value of the execution time and makespan of these jobs on each VM is determined after the completion of their corresponding tasks by VM_1 and VM_5 before and after live and storage migration. These values are calculated on the basis of the logic explained in Section 6.3.1 and summarized in Table 11.

As can be seen from the results, 7:55:46 hours in total is taken by VM_1 and VM_5 to execute the jobs with 2:41:06 total makespan when storage migration is applied. In live migration, the jobs are completed after 2:38:16 when 7:48:25 hours in total has been taken for their execution.

Table 11. Implementation results using the VMware ALB

VM ID	Location	Scheduled Tasks	Job Execution Time	Job Makespan
		VM storage migration		

1	PM_1Before migration	$t_5, t_{11}, t_{41}, t_{45}$	6:51:06	2:05:49
	PM_1 After migration	$t_5, t_{11}, t_{41}, t_{45}, t_{47}, t_{49}, t_{30}, t_{21}, t_{25}, t_{27}, t_{50}$		
5	PM_1Before migration	t_{44}, t_{46}	1:04:40	0:35:17
	PM_2 After migration	$t_{46}, t_{46}, t_{51}, t_{27}, t_{39}$		
Total			**7:55:46**	**2:41:06**
		VM live migration		
1	PM_1Before migration	$t_5, t_{11}, t_{41}, t_{45}$	6:46:45	2:04:30
	PM_1 After migration	$t_5, t_{11}, t_{41}, t_{45}, t_{47}, t_{49}, t_{30}, t_{21}, t_{25}, t_{27}, t_{50}$		
5	PM_1Before migration	t_{44}, t_{46}	1:01:40	0:33:46
	PM_2 After migration	$t_{46}, t_{46}, t_{51}, t_{27}, t_{39}$		
Total			**7:48:25**	**2:38:16**

6.4. Evaluation Results Analyses

The final results of implementing the two approaches are summarized in Table 12.

Table 12. Comparison of results

	Time (Seconds and Hours)					Power (kw)	Memory (MB)		
	MOTS-PSO time	Total execu-tion time	Total makespan (Max Path)	Task transfer time	VM migration time		VM transferred memory	Task size	Transferred data
VM Storage Migration	NA	28546 (7:55:46)	9666 (2:41:16)	NA	330 (0:5:30)	31.71	297059	0.1536	0
VM Live Migration	NA	28102 (7:48:25)	9486 (2:38:06)	NA	3 (0:0:3)	31.22	100	0.1536	0
MO-LB	0.5 (00:00:0.5)	27569 (7:39:29)	5640 (1:34:00)	0	NA	30.63	NA	0.1536	0

When the ALB was applied in the defined scenario, VM_5 was migrated after PM_1 became overloaded and its allocated VMs experienced tension and low performance for a period of time. This affected the execution time of tasks on both VMs and it took longer for these VMs to complete their jobs. In contrast, the amount of execution time with the application MO-LB was reduced by 3.4% and 1.9% compared to VM storage and live migration respectively, as a result of executing tasks by VMs on another PM with lower resource utilization. In addition, makespan was reduced dramatically (41.6% and 40.5% compared to VM storage and live migration respectively) by MO-LB, as the tasks were distributed over eight VMs, and this method benefits from parallel task execution. Furthermore, 100 Mb more memory was transferred for the ALB during live VM migration compared to the MO-LB system. The ALB also consumed 297059 Mb ($\approx$ 290 GB) more memory on both the original and destination PMs and data-stores for 5.5 minutes compared to the MO-LB system, and used more bandwidth due to VM migration.

The additional amount of power consumed per hour by an active CPU can be estimated using Equation 19 where $pw_0 = 40kw/h$ and $\alpha = 0.1 * pw_0$ (see Section 5.1.3):

$$Pw_{active}(CPU) = (pw_0 * 0) + [(0.1 * pw_0) * 1] = 4 * 1 = 4kw/h \quad (41)$$

In this study it is assumed that the total execution time is the summation of time taken by each task to reach completion and that each task applies one CPU during execution. Therefore, it can be assumed that the power of only one CPU is applied during the total execution time of tasks. Using this calculation, the additional amount of power consumed by applying the ALB using storage migration to execute all scheduled tasks (T_{set}) —which takes 28546 seconds or 7.93 hours—is:

$$Additional\_Pw_{active}^{ALB-StorageM}(T_{set}) = 4kw/h * 7.93h = 31.71kw \quad (42)$$

The additional amount of power consumed by the ALB using live migration to execute T_{set} during 28102 seconds (7.80 hours) is also calculated as:

$$Additional\_Pw_{active}^{ALB-LiveM}(T_{set}) = 4kw/h * 7.80h = 31.22kw \quad (43)$$

In the MO-LB system, the number of activated PMs was not changed because the model applies VMs on active PMs. Therefore, the additional power consumption by the MO-LB system for executing T_{set} —which takes 27569 seconds or 7.65 hours—can be estimated as follows:

$$Additional\_Pw_{active}^{FP-TBSLB}(T_{set}) = 4kw/h * 7.65h = 30.63kw \quad (44)$$

This shows that the proposed method has the lowest power consumption.

In addition, it takes 330 seconds for the ALB to perform VM storage migration, and three seconds for VM live migration. In contrast, the MO-LB system achieves load balancing in at most 0.5 second, which is the time taken by the MOTS-PSO algorithm to find the optimal scheduling pattern. The task transfer time taken in the MO-LB system is near zero. The time taken for load balancing is reduced by 99.8% and 83.3% using the MO-LB system compared to VM storage and live migration respectively.

In summary, the MO-LB system achieves lower execution time, job makespan, memory transfer, bandwidth, and power consumption compared to live and storage migration strategies applied by VMware-ESXi. In addition, the MO-LB system takes less time to conduct load balancing over PMs compared to the ALB of VMware-ESXi.

7. Conclusion and Future Works

Several VM migration techniques have been applied for load balancing and optimizing resource utilization in cloud environments, including suspension/resumption and live migration. Live migration has been developed to migrate running VMs, and achieves lower VM downtime compared to the suspend/resume strategy. However, live migration for large VMs is not an optimal solution because it consumes time, bandwidth, power, and memory space in both origin and destination PMs and data-stores. In addition, the live migration process carries the risk of losing last user activities.

In this study, an MO-LB system has been proposed instead of VM migration for load balancing. The MO-LB system distributes accumulated tasks in the task queue of the primary VMs over a set of compatible VMs with lower utilization. This method includes a CPU Utilization Prediction (CUP) method that not only forecasts primary VMs, but also determines a set of VMs as candidate destinations for arrival tasks to primary VMs. A Multi-Objective Task Scheduling optimization model using Particle Swarm Optimization (MOTS-PSO) was designed to transfer tasks to the selected destination VMs, minimizing task transfer time, task execution cost/time, length of VM task queue, and power consumption. In addition, the primary VMs were not slowed down by the application of the MO-LB system. We evaluated the proposed model by applying a VMware-vSphere based private cloud in comparison with the ALB of VMware-ESXi which applies VM live and storage migration. The evaluation results show that the MO-LB system achieved a reduction in makespan, execution time, memory usage, power consumption, and total time taken for load balancing compared to the ALB in both live and storage VM migration. As a result, the proposed method dramatically increased VM performance and reduced service response time. This method can be applied in the hypervisor layer to optimize resource management and load balancing, boosting the Quality of Service (QoS) expected by cloud customers.

When the CPU speed of the destination VM is lower than the CPU speed of the primary VM, however, the task execution time and consequently power consumption will be increased. In our future work, therefore, we will add extra constraints to our MOTS-PSO problem to consider scheduling tasks to VMs that have a CPU speed equal to or more than the CPU speed of the primary VM. Using this strategy, there would not be even a slight increase in total execution time and power consumption. In addition, the solution will be improved to cover scheduling scientific workflows (such as Pegasus applications), and the application of this solution to solve the problem of cold spot PMs will be evaluated. This future study will also consider heterogeneous cloud environments as its research direction, and the prediction sub-system

of the developed model will be improved. We will also consider different amounts of CPU utilization by task (tcu_i) to obtain a better estimation of task execution time.

Acknowledgment

The work presented in this paper was supported by the Australian Research Council (ARC) under Discovery Project DP140101366.

References

[1] Celesti, A., Fazio, M., Villari, M. and Puliafito, A. (2012) Virtual machine provisioning through satellite communications in federated Cloud environments. *Future Generation Computer Systems*, **28**(1), 85-93.
[2] Buyya, R., Broberg, J. and Goscinski, A. (2011) Cloud Computing, Principles and Paradigms. John Wiley & Sons, Hoboken, NJ.
[3] Sallam, A. and Li, K. (2014) A multi-objective virtual machine migration policy in cloud systems. *The Computer Journal*, **57**(2), 195-204.
[4] Jin, H., Gao, W., Wu, S., Shi, X., Wu, X. and Zhou, F. (2011) Optimizing the live migration of virtual machine by CPU scheduling. *Journal of Network and Computer Applications*, **34**(4), 1088-1096.
[5] Kozuch, M. and Satyanarayanan, M. (2002) Internet suspend/resume. *Proceedings of the 4th IEEE Workshop on Mobile Computing Systems and Applications*, Callicoon, USA, 20-21 June, pp. 40-46. IEEE- Los Alamitos, California.
[6] Sapuntzakis, C. P., Chandra, R., Pfaff, B., Chow, J., Lam, M. S. and Rosenblum, M. (2002) Optimizing the migration of virtual computers. *Proceedings of the 5th symposium on Operating systems design and implementation*, Boston, USA, 8-11 December, pp. 377-390. ACM- New York, NY, USA.
[7] Whitaker, A., Cox, R. S., Shaw, M. and Gribble, S. D. (2004) Constructing services with interposable virtual hardware. *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, USA, 29-31 March, pp. 169-82. ACM- New York, NY, USA.
[8] Hines, M. R. and Gopalan, K. (2009) Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. *Proceedings of Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, Washington, DC, USA, 11-13 March, pp. 51-60. ACM- New York, NY, USA.
[9] Barker, S., Chi, Y., Moon, H. J., Hacigümüş, H. and Shenoy, P. (2012) Cut me some slack: Latency-aware live migration for databases. *Proceedings of the 15th international conference on extending database technology*, Berlin, Germany, 27-30 March, pp. 432-443. ACM- New York, NY, USA.
[10] Cecchet, E., Singh, R., Sharma, U. and Shenoy, P. (2011) Dolly: virtualization-driven database provisioning for the cloud. *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, Newport Beach, USA, 09-11 March, pp. 51-62. ACM- New York, NY, USA.
[11] Elmore, A. J., Das, S., Agrawal, D. and El Abbadi, A. (2011) Zephyr: live migration in shared nothing databases for elastic cloud platforms. *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, Athens, Greece, 12-16 June, pp. 301-312. ACM- New York, NY, USA.
[12] Sakr, S. and Liu, A. (2012) SLA-based and consumer-centric dynamic provisioning for cloud databases. *Proceedings of 5th IEEE International Conference on Cloud Computing (CLOUD)*, Honolulu, USA, 24-29 June, pp. 360-367. IEEE- California, USA.
[13] Jain, N., Menache, I., Naor, J. and Shepherd, F. (2012) Topology-aware VM migration in bandwidth oversubscribed datacenter networks. *Proceedings of 39th International Colloquium on Autonomic, Language, and Programming*, Warwick, UK, July 9-13, pp. 586-597. Springer- Verlag Berlin, Heidelberg.
[14] Osman, S., Subhraveti, D., Su, G. and Nieh, J. (2002) The design and implementation of ZAP: A system for migrating computing environments. *ACM SIGOPS Operating Systems Review*, **36**(SI), 361-376.
[15] Jun, C. and Xiaowei, C. (2011) IPv6 virtual machine live migration framework for cloud computing. *Energy Procedia*, **13,** 5753-5757.
[16] Nelson, M., Lim, B. H. and Hutchins, G. (2005) Fast transparent migration for virtual machines. *Proceedings of USENIX Annual Technical Conference*, Anaheim, USA, 10-15 April, pp. 391-394. USENIX Association- Berkeley, CA, USA.
[17] Clark, C., Fraser, K., Hand, S. and Jacob, G. H. (2005) Live migration of virtual machines. *Proceedings of the 2nd ACM/USENIX Symposium on Network Systems, Design and Implementation (NSDI)*, 2-4 May, pp. 273-286. USENIX Association- Berkeley, CA, USA.
[18] Nicolae, B. and Cappello, F. (2013) BlobCR: Virtual disk based checkpoint-restart for HPC applications on IaaS clouds. *Journal of Parallel and Distributed Computing*, **73**(5), 698-711.
[19] Liao, X., Jin, H. and Liu, H. (2012) Towards a green cluster through dynamic remapping of virtual machines. *Future Generation Computer Systems*, **28**(2), 469-477.

[20] Lin, W., Wang, J. Z., Liang, C. and Qi, D. (2011) A threshold-based dynamic resource allocation scheme for cloud computing. *Procedia Engineering*, **23,** 695 – 703.
[21] Atif, M. and Strazdins, P. (2014) Adaptive parallel application resource remapping through the live migration of virtual machines. *Future Generation Computer Systems*, **37,** 148-161.
[22] Ramezani, F., Lu, J. and Hussain, F. K. (2013) Task-Based System Load Balancing in Cloud Computing Using Particle Swarm Optimization. *International Journal of Parallel Programming*, **42**(5), 739-754.
[23] Mishra, M., Das, A., Kulkarni, P. and Sahoo, A. (2012) Dynamic resource management using virtual machine migrations. *Communications Magazine, IEEE*, **50**(9), 34-40.
[24] Forsman, M., Glad, A., Lundberg, L. and Ilie, D. (2015) Algorithms for automated live migration of virtual machines. *Journal of Systems and Software*, **101,** 110-126.
[25] (2015) HTCondor™ Version 8.3.8 Manual. Center for High Throughput Computing, U. o. W.-M., Madison, USA.
[26] (2016) Pegasus 4.6.2 User Guide. California, U. o. S., Santa Monica, USA.
[27] Islam, S., Keung, J., Lee, K. and Liu, A. (2012) Empirical prediction models for adaptive resource provisioning in the cloud. *Future Generation Computer Systems*, **28**(1), 155-162.
[28] Yang, D., Cao, J., Fu, J., Wang, J. and Guo, J. (2013) A pattern fusion model for multi-step-ahead CPU load prediction. *Journal of Systems and Software*, **86**(5), 1257-1266.
[29] Ramezani, F., Lu, J., Taheri, J. and Hussain, F. (2015) Evolutionary algorithm-based multi-objective task scheduling optimization model in cloud environments. *World Wide Web*, **18**(6), 1-21.
[30] Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A. F. and Buyya, R. (2011) CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, **41**(1), 23-50.
[31] Cirne, W., Brasileiro, F., Andrade, N., Costa, L. B., Andrade, A., Novaes, R. and Mowbray, M. (2006) Labs of the world. *Journal of Grid Computing*, **4**(3), 225-246.
[32] Tchernykh, A., Pecero, J. E., Barrondo, A. and Schaeffer, E. (2014) Adaptive energy efficient scheduling in peer-to-peer desktop grids. *Future Generation Computer Systems*, **36**(0), 209-220.
[33] Shieh, W.-Y. and Pong, C.-C. (2013) Energy and transition-aware runtime task scheduling for multicore processors. *Journal of Parallel and Distributed Computing*, **73**(9), 1225-1238.
[34] Wang, X., Wang, Y. and Cui, Y. (2014) A new multi-objective bi-level programming model for energy and locality aware multi-job scheduling in cloud computing. *Future Generation Computer Systems*, **36**(0), 91-101.
[35] Priya, B., Pilli, E. S. and Joshi, R. C. (2013) A survey on energy and power consumption models for Greener Cloud. *Proceedings of IEEE 3rd International Conference on Advanced Computing (IACC)*, 22-23 February, pp. 76-82. IEEE.
[36] Zhang, Y.-W. and Guo, R.-F. (2013) Power-aware scheduling algorithms for sporadic tasks in real-time systems. *Journal of Systems and Software*, **86**(10), 2611-2619.
[37] Buyya, R., Beloglazov, A. and Abawajy, J. (2010) Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges. *Proceedings of the 2010 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2010)*, Las Vegas, USA, 12-15 July, pp. 1-12. arXiv preprint.
[38] Gao, Y., Guan, H., Qi, Z., Hou, Y. and Liu, L. (2013) A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. *Journal of Computer and System Sciences*, **79**(8), 1230-1242.
[39] (2008) Altix® XE320 System User's Guide. Silicon Graphics International Corp.
[40] Top 500 Supercomputing Sites (2014) http://www.top500.org/system/176223
[41] Guo, L., Zhao, S., Shen, S. and Jiang, C. (2012) Task Scheduling Optimization in Cloud Computing Based on Heuristic Algorithm. *Journal of Networks*, **7**(3), 547-553.
[42] Lei, Z., Yuehui, C., Runyuan, S., Shan, J. and Bo, Y. (2008) A task scheduling algorithm based on PSO for grid computing. *International Journal of Computational Intelligence Research*, **4**(1), 37-43.
[43] Ramezani, F., Lu, J. and Hussain, F. (2012) Tasks based system load balancing approach in cloud environment. *Proceedings of International Conference on Intelligent Systems and Knowledge Engineering*, Beijing, China, 15-17 December, pp. 31-42. Springer - Berlin Heidelberg.
[44] Salman, A., Ahmad, I. and Al-Madani, S. (2002) Particle swarm optimization for task assignment problem. *Microprocessors and Microsystems*, **26**(8), 363-371.
[45] Liu, H., Abraham, A., Snášel, V. and McLoone, S. (2012) Swarm scheduling approaches for work-flow applications with security constraints in distributed data-intensive computing environments. *Information Sciences*, **192**(0), 228-243.
[46] Mahmoodabadi, M. J., Bagheri, A., Nariman-zadeh, N. and Jamali, A. (2012) A new optimization algorithm based on a combination of particle swarm optimization, convergence and divergence operators for single-objective and multi-objective problems. *Engineering Optimization*, **44**(10), 1-20.
[47] Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T. (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, **6**(2), 182-197.
[48] Alves, M. J. (2012) Using MOPSO to solve multiobjective bilevel linear problems. *Proceedings of International Conference on Swarm Intelligence*, Shenzhen, China, 17-20 June, pp. 332-339. Springer- Verlag Berlin, Heidelberg.
[49] Gao, Y., Zhang, G., Lu, J. and Wee, H.-M. (2011) Particle swarm optimization for bi-level pricing problems in supply chains. *Journal of Global Optimization*, **51**(2), 245-254.
[50] Lu, J., Zhang, G. and Ruan, D. (2007) Multi-objective Group Decision Making: Methods, Software and Applications with Fuzzy Set Techniques. Imperial College Press, London.

[51] Naderpour, M., Lu, J. and Zhang, G. (2014) An intelligent situation awareness support system for safety-critical environments. *Decision Support Systems*, **59**, 325-340.
[52] Kennedy, J. and Eberhart, R. (1995) Particle swarm optimization. *Proceedings of The IEEE International Conference on Neural Networks*, Perth, Australia, pp. 1942-1948. IEEE Service Center- Piscataway, NJ.
[53] Engelbrecht, A. P. (2006) Fundamentals of Computational Swarm Intelligence. John Wiley & Sons, Hoboken, NJ.
[54] Engelbrecht, A. P. (2007) Computational intelligence: an introduction. John Wiley & Sons, Ltd, Hoboken.
[55] Poli, R., Kennedy, J. and Blackwell, T. (2007) Particle swarm optimization. *Swarm Intelligence*, **1**(1), 33-57.
[56] Tasgetiren, M. F., Sevkli, M., Liang, Y.-C. and Gencyilmaz, G. (2004) Particle swarm optimization algorithm for single machine total weighted tardiness problem. *Proceedings of The Congress on Evolutionary Computation (CEC2004)*, 19-23 June, pp. 1412-1419. IEEE.
[57] Dubey, I. and Gupta, M. (2015) Enhanced Particle Swarm Optimization with Uniform Mutation and SPV Rule for Grid Task Scheduling. *International Journal of Computer Applications*, **116**(15), 14-17.
[58] Kumar, N. and Vidyarthi, D. P. (2016) A novel hybrid PSO–GA meta-heuristic for scheduling of DAG with communication on multiprocessor systems. *Engineering with Computers*, **32**(1), 35-47.
[59] Zhang, L., Chen, Y., Sun, R., Jing, S. and Yang, B. (2008) A task scheduling algorithm based on PSO for grid computing. *International Journal of Computational Intelligence Research*, **4**(1), 37-43.
[60] Barati, M. and Sharifian, S. (2015) A hybrid heuristic-based tuned support vector regression model for cloud load prediction. *Journal of Supercomputing*, **71**(11), 4235-4259.