Evolutionary Algorithm-based Multi-Objective Task Scheduling Optimization Model in Cloud Environments

Fahimeh Ramezani[a], Jie Lu[a], Javid Taheri[b], Farookh Khadeer Hussain[a]

[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] Center for Distributed and High Performance Computing, School of Information Technologies, University of Sydney, NSW 2006, Australia

{Fahimeh.Ramezani,Jie.Lu,Farookh.Hussain}@uts.edu.au

javid.taheri@sydney.edu.au

Abstract. Optimizing task scheduling in a distributed heterogeneous computing environment, which is a nonlinear multi-objective NP-hard problem, plays a critical role in decreasing service response time and cost, and boosting Quality of Service (QoS). This paper, considers four conflicting objectives, namely minimizing task transfer time, task execution cost, power consumption, and task queue length, to develop a comprehensive multi-objective optimization model for task scheduling. This model reduces costs from both the customer and provider perspectives by considering execution and power cost. We evaluate our model by applying two multi-objective evolutionary algorithms, namely Multi-Objective Particle Swarm Optimization (MOPSO) and Multi-Objective Genetic Algorithm (MOGA). To implement the proposed model, we extend the Cloudsim toolkit by using MOPSO and MOGA as its task scheduling algorithms which determine the optimal task arrangement among VMs. The simulation results show that the proposed multi-objective model finds optimal trade-off solutions amongst the four conflicting objectives, which significantly reduces the job response time and makespan. This model not only increases QoS but also decreases the cost to providers. From our experimentation results, we find that MOPSO is a faster and more accurate evolutionary algorithm than MOGA for solving such problems.

Keywords: Cloud computing, Task Scheduling, Multi-Objective Particle Swarm Optimization, Multi-Objective Genetic Algorithm, Jswarm, Cloudsim

1 Introduction

Cloud computing is a service-oriented computing paradigm that has significantly revolutionized computing by offering three web-based services – Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) [1]. SaaS and PaaS users generate several jobs in a cloud environment. To deliver these services over the World Wide Web, jobs generated by users are submitted to schedu-

lers to be executed by a set of processors (cloud resources). Each job consists of several dependent tasks described by a Directed Acyclic Graph (DAG)[2]. In the cloud environment, the number of tasks in a workflow, as well as the number of available resources, can grow quickly, especially when virtual resources are allocated. Calculating all possible task-resource mappings in the cloud environment and selecting the optimal mapping is not feasible, since the complexity would grow exponentially with the number of tasks and resources [3]. Therefore, the development of intelligent task scheduling mechanisms that take into account the efficiency of all the cloud computing facilities, has become a critical part of current cloud optimization problems and plays a key role in improving flexible and reliable systems. The main purpose is to schedule tasks to adaptable resources in accordance with time, which involves establishing a proper sequence whereby tasks can be executed under transaction logic constraints [4].

There are several studies [1, 3-10] that mainly emphasize the minimization of job makespan and task execution cost in their multi-objective optimization models by applying evolutionary algorithms. However, these studies fail to consider the need to minimize power consumption by the cloud infrastructure. Meanwhile, there is a considerable amount of research work that focuses on reducing power consumption in their proposed bi-objective task scheduling models by minimizing the value of their developed predefined power consumption objective functions for multi-core processors and cloud environment [11, 12]. In addition, several studies have been undertaken in the area of energy-aware task scheduling by applying the Dynamic Voltage Frequency Scaling (DVFS) technique [13-16].

To the best of our knowledge, reducing the task queue length of VMs has not been investigated in proposed task scheduling optimization models. In addition, our work is the first to minimize task transfer time, task execution cost, power consumption, and task queue length concurrently in a task scheduling optimization model. A longer task queue results in more waiting time for tasks and a longer response time. In current task scheduling optimization models that apply evolutionary algorithms with predefined objective functions, the optimal task scheduling schemas are suggested based on task and VM properties. In such models, optimizing objective functions usually occurs by assigning tasks to high performance VMs and neglecting low performance VMs. This leads to the creation of a long task queue for some VMs, while some other VMs remain idle. Although the properties of low performance VMs do not optimize the value of objective functions, as they have the lowest number of CPUs and smallest amount of memory, they can decrease response time by executing waiting tasks in the queues.

To improve previous research work and address this shortcoming in the existing literature, a multi-objective model for task scheduling in a cloud environment that considers four aspects of optimizing cloud utilization is proposed in this study. The model aims to enhance QoS based on the points of view of both cloud users and providers by minimizing service response time and price (to raise customer satisfaction), and minimizing power consumption (to reduce providers' expenditure). In addition, we determine an objective function to achieve optimal load balance between resources and control the length of task queues in a cluster. This objective function

considers changes in resource capacity (memory and number of CPUs) to avoid assigning multiple tasks to one VM's processors and creating long task queues. To find the optimal solution for the proposed model, an algorithm is developed based on Multi-Objective Particle Swarm Optimization (MOPSO) and Multi-Objective Genetic Algorithm (MOGA). Cloudsim toolkit [17] is extended by applying MOPSO and MOGA as its task scheduling algorithms to implement and evaluate the proposed model.

This paper focuses on scheduling highly parallel computations such as Bag-of-Tasks (BoT) applications for SaaS in a cloud environment. In such applications, 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 [18, 19].

The efficiency of the proposed model is evaluated through different scenarios. Simulation results show that the proposed model significantly increases QoS by considering more criteria for optimization, and has the ability to satisfy both users and providers. In fact, the proposed model is able to determine trade-off solutions that offer the best possible compromises among the optimization objectives, and not only helps cloud providers to reduce the cost of power consumed, but also helps them to maintain the expected level of QoS. It has also been found that MOPSO is a faster and more accurate evolutionary algorithm than MOGA for solving such problems. The contributions of the paper are summarized as follows:

1) Develop a multi-objective task scheduling model to minimize task transfer time, task execution cost, power consumption, and task queue length.
2) Develop intelligent methods to estimate: (1) task transfer time, (2) task execution cost, (3) power consumption based on the number of active Physical Machines (PMs), and (4) task queue length based on the remaining resource capacity (memory and idle CPUs).
3) Develop a MOPSO/MOGA-based algorithm to solve the proposed multi-objective task scheduling problem, and compare two evolutionary algorithms in different scenarios.
4) Extend the Cloudsim package to evaluate the model using MOPSO and MOGA.

The rest of the paper is organized as follows. In Section 2, related works are presented. Section 3 illustrates a multi-objective model for task scheduling optimization. Section 4 presents a developed MOPSO/MOGA-based algorithm to determine the optimal solution (task scheduling pattern) for the proposed multi-objective model. The model is evaluated in Section 5. Lastly, the conclusion and future works are provided in Section 6.

2 Related Works

The task scheduling problem in distributed computing systems is an NP-hard optimization problem that also affects QoS in the cloud environment by optimizing ser-

vice cost and service response time. Therefore, the use of a heuristic algorithm ensures an acceptable runtime of the scheduling algorithm itself since it significantly reduces the complexity of the search space. In this regard, Song et al. [5] proposed a general job selection and allocation framework that utilizes an adaptive filter to select jobs and a modified heuristic algorithm (Min-Min) to allocate them. Two objectives—maximizing the remaining CPU capacity, and the utilization of resources—and four criteria—the resource requirements of CPU, memory, hard-disk and network bandwidth—were considered for the optimization problem. Li et al. [8] applied an Ant Colony optimization approach to create a better scheduling result with shorter total-task-finish time and mean-task finish time. Li et al. [6] took resource allocation patterns into account and proposed a task and resource optimization mechanism. Tayal [4] proposed a fuzzy-GA based optimization approach to enhance the accuracy of GA results for the job scheduling process, which makes a scheduling decision by evaluating the entire group of tasks in the job queue. Juhnke et al. [3] proposed a multi-objective scheduling algorithm for cloud-based workflow applications to minimize cost and execution time by applying the Pareto Archived Evolution Strategy, which is a type of GA. Lei et al. [9] and Salman et al. [7] developed an optimization model to optimize task execution time, and showed that the particle swarm optimization (PSO) algorithm is able to obtain a better schedule than GA in grid computing and distributed systems. Guo [10] also proposed a multi-objective task scheduling model to minimize task execution time and cost using the PSO algorithm. Taheri et al. [1] considered the data-files required for jobs from public or private clouds and proposed a bi-objective job scheduling optimization model to minimize job execution and data file transfer time using PSO. These studies mainly emphasized the minimization of the job makespan and task execution cost in their multi-objective optimization models by applying evolutionary algorithms, and the reduction of power consumption is neglected in such studies.

Meanwhile, there are lots of works focus on reducing power consumption in their proposed bi-objective task scheduling models by minimizing the value of their developed predefined power consumption objective functions for multi-core processors and cloud environment [11, 12]. Different objective functions have been suggested in these models for estimating energy consumption [19-21]. Shieh and Pong [11] designed an energy- and transition-aware algorithm to schedule periodic tasks in multi-core systems considering voltage transition overheads. They suggested an integer linear programming model to find the optimal power-aware scheduling for specific task set and core number. Wang et al. [12] developed an energy-aware multi-objective bi-level programming model based on MapReduce. They considered energy consumption in the data placement process, combined with a local multi-job scheduling scheme. Their approach has three steps. First, they considered changes in energy consumption along with the performance of servers. Then, their model dynamically adjusts data locality based on the current network state. In the last step, they developed an integer bi-level programming model considering the fact that task-scheduling schemas depend on data placement patterns.

Several investigations have also been conducted into developing energy-aware task scheduling algorithms to optimize energy consumption by applying Dynamic

Voltage Scaling (DVS) technique [13-16]. In these works, the authors determined task-slack time by considering critical and non-critical paths in job DAGs. They then extended non-critical task execution times by scaling down the voltage frequency of corresponding processors. In other works, energy is reduced by sending non-critical tasks to the most cost efficient VMs [15].

Although a significant number of studies have been carried out in the area of power consumption reduction in a cloud/grid environment, the reduction of task queue length in a cluster by considering the workload capacity of VMs has been neglected in earlier multi-objective task scheduling models. Task queue length is an effective factor for reducing job makespan because it determines the wait and finish time of tasks. In addition, the following four optimization objectives: (1) task transfer time, (2) task execution cost, (3) power consumption, and (4) the task queue length of VMs, have not been considered in the previous works to develop a comprehensive task scheduling model.

3 Problem Formulation

A cloud environment dynamically receives a large number of tasks from its applications' users in every portion of each second. These tasks accumulate in several queues and are then sent to the task schedulers. The task schedulers are responsible for allocating these tasks to VMs for execution. These VMs which are in turn allocated to PMs (see Fig. 1), have different numbers of virtual CPUs and different memory size. The task schedulers apply optimization procedures to allocate task among VMs to achieve optimal resource utilization in a cloud environment. The scheduling process repeats dynamically to schedule every set of arrival tasks among VMs. Considering this fact that independent users arbitrarily send tasks to cloud environment, the number and type of tasks in each queue may significantly change from one scheduling to another.

To create a higher level of resource utilization while minimizing cost and maximizing QoS, we develop a multi-objective model for optimizing task scheduling that considers four aspects of the task scheduling optimization problem: task execution cost, task transfer time, task queue length and power consumption. The variables that are applied to formulate this multi-objective model are defined in Table 1.

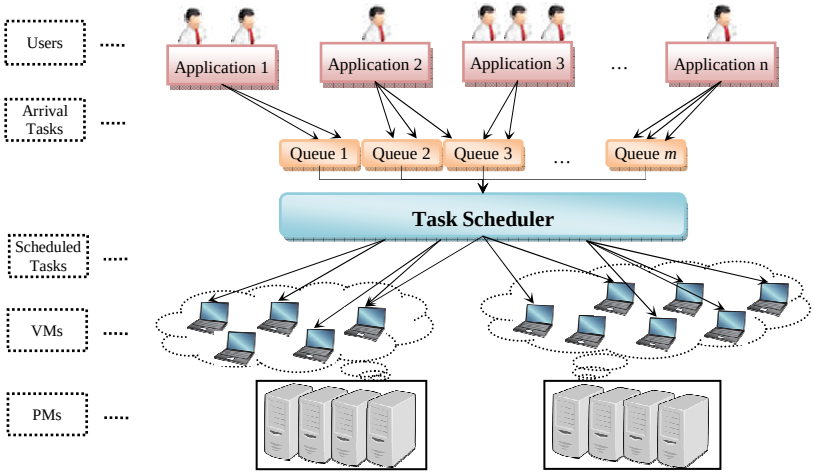

Figure 1. Cloud objects and their relations

Table 1. Definitions of Variables

Symbol	Definition
n	The number of arrival tasks
T	The set of arrival tasks = $\{t_1, t_2, \ldots, t_n\}$
DE_{ik}	The amount of data (MB) that task i assigns to VM_k to be executed = ($task_i$ data length + $task_i$ output size)
DT_{ik}	The amount of data (MB) needing to be transferred = $task_i$ file size
tm_j	The amount of memory (MB) required for executing task j
tc_j	The number of CPUs required for executing task j
$Texe_k$	The total task execution time on VM_k
m	The number of VMs
VM_j	Virtual Machine j, j={1, 2, …, m}
VMm_k	The amount of memory of VM_k (MB ≈ 0.001 GB)
VMc_k	The number of CPUs in VM_k
N^k_{aCPU}	The number of active CPUs on VM_k
$VMbw_k$	The bandwidth of VM_k (Mb/s)
VM^k_{mips}	VM computing speed (Million Instructions Per Second)
$RVMm^k$	The amount of available memory on VM_k
$RVMc^k$	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	The set of VMs located on the zth PM = $\{k \mid VM_k \in z\text{th PM}, z \in \{1,2,\ldots,N_{PM}\}\}$
cp	The number of cloud providers
C_p	Maximum capacity for provider p
SP_p	The set of VMs belonging to pth provider = $\{k \mid VM_k \in P\text{th Cloud provider}, P \in \{1,2,\ldots,cp\}\}$
$Pcost_p$	The cost of one unit CPUs for pth provider (AUD/hour)
x_{ij}	1 if task i is assigned to VM_k and 0, otherwise

*Note: Task o*utput size is the amount of data that each task produces during its execution and need to be executed by its subsequent tasks

3.1 Tasks Transfer Time

We apply and improve the formula proposed in [10] to estimate the total task transfer time as follows:

$$Ttrans = \sum_{k=1}^{m}\sum_{i=1}^{n} x_{ik} * \frac{DT_{ik}}{VMbw_k * (\frac{1}{8})} \quad (1)$$

The coefficient 1/8 is used to convert Mb to MB.

3.2 Task Execution Cost

In this paper, the task execution cost (AUD per hour) for provider p is calculated as follows:

$$Cexe_p = \sum_{k \in SP_p} Pcost_p * Texe_k \quad (2)$$

where, $Texe_k$ is the estimated execution time of assigned tasks to VM_k in hour, and $Pcost_p$ is the cost of one unit CPUs for pth provider in AUD per hour. In this formula, the total task execution time in each VM belonging to provider p is calculated. This value is then multiplied by the provider's price for using unit CPU.

Total task execution time in VM_k is calculated as follows:

$$Texe_k = \frac{1}{3600} * \sum_{i=1}^{n} x_{ik} * \frac{DE_{ik}}{\mathrm{VM}_{\mathrm{mips}}^{\mathrm{k}}} \quad (3)$$

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.

Finally, the total task execution cost for all providers is determined as:

$$Cexe = \sum_{p=1}^{cp} Cexe_p \quad (4)$$

3.3 Power Consumption

Reducing power consumption in large scale computing systems such as grid and cloud environments has been investigated as an important contributor to the reduction of operating costs and diverse environmental impacts.

Several researches have been completed in the area of power consumption reduction through the proposal of power-aware multi-objective task scheduling models for

multi-core processors, grid and cloud environments [11, 12]. In these models, a variety of linear and non-linear objective functions have been suggested to estimate power consumption based on task scheduling patterns [19-21]. All these models have been developed based on the fact that the energy will be reduced when the PM is either off or in idle mode. It has also been proved by Buyya et al. [20, 22] that an idle server consumes around 70% of the power compared to a fully utilized server. In a nutshell, previous studies show that having fewer active CPUs and PMs leads to lower power consumption in a cluster.

Considering this fact, instead of estimating total energy consumption—which can be exactly determined in a real cluster after scheduling and executing tasks— in this study we minimize the ratio of active PMs and CPUs to all available PMs and CPUs to reduce power consumption. To do this, the optimization model avoids choosing VMs on idle PMs as destinations for scheduled tasks and consequently reduces consumed power in the corresponding cloud cluster. Equation 5 is then developed based on this theory as an objective function for our optimization model. Using this, the percentage of active PMs and CPUs is determined as a measure for reducing power consumption as:

$$PowerC = \frac{(N_{aPM} + \mu \sum_{k=1}^{m} N_{aCPU}^{k})}{(N_{PM} + \mu \sum_{k=1}^{m} \mathrm{VMc_k})} \tag{5}$$

where $\mathrm{N_{aCPU}^{k}}$ is the number of active CPUs in $\mathrm{VM_k}$. To calculate $\mathrm{N_{aCPU}^{k}}$ we consider the fact that the number of tasks assigned to $\mathrm{VM_k}$ may be less than the number of CPUs in $\mathrm{VM_k}$, or may exceed the maximum CPU number in this VM. Therefore, $\mathrm{N_{aCPU}^{k}}$ is equal to the minimum value between the total number of tasks assigned to $\mathrm{VM_k}$ and the total number of CPUs in this VM as:

$$N_{aCPU}^{k} = \min\left(\sum_{i=1}^{n} x_{ik}, \mathrm{VMc_k}\right) \tag{6}$$

The number of active PMs in all iterations ($\mathrm{N_{aPM}}$) is estimated based on the number of active VMs on each PM. The number of activated VMs is calculated based on the number of tasks assigned to them. If at least one task assigned to $\mathrm{VM_k}$ is located on $\mathrm{PM_z}$, this VM will be activated, thus its corresponding PM should be turned on and activated as well. As a result, the number of active PM is determined using the following formula:

$$N_{aPM} = \sum_{k \in Svm_z} \min\left(\sum_{i=1}^{n} x_{ik}, 1\right) \tag{7}$$

where $\sum_{i=1}^{n} x_{ik}$ is the number of tasks assigned to CPUs of $\mathrm{VM_k}$.

We also prove that minimizing Equation 5 reduces power consumption in the cloud cluster. To achieve this, we firstly determine power consumption for each fully utilized PM as follows based on this fact that the power consumed by PM increase by the number of its active CPUs:

$$Pw_{ful}(PM_z) = Pw_0^z + \alpha\ N_{CPU}^z \tag{8}$$

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

$$\alpha = \frac{Pw_{ful}(PM_z) - Pw_0^z}{N_{CPU}^z} \tag{9}$$

Since we have homogenous PMs in our cluster, all the PMs in this cluster have the same value for α and Pw_0. Therefore, the total amount of consumed power in this cluster can be determined as follows:

$$Pw_{ful}(Cluster) = \sum_{z=1}^{N_{PM}} Pw_{ful}(PM_z) = \sum_{z=1}^{N_{PM}} Pw_0^z + \alpha \sum_{z=1}^{N_{PM}} N_{CPU}^z \tag{10}$$

then

$$Pw_{ful}(Cluster) = Pw_0^z * N_{PM} + \alpha * N_{CPU} \tag{11}$$

where N_{CPU} is the total number of CPUs in the cluster. On the other hand, $N_{CPU} = \sum_{k=1}^{m} \mathrm{VMc_k}$. Using this $Pw_{ful}(Cluster)$ is calculated as:

$$Pw_{ful}(Cluster) = Pw_0^z * N_{PM} + \alpha * \sum_{k=1}^{m} \mathrm{VMc_k} \tag{12}$$

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

$$Pw_{active}(Cluster) = Pw_0^z * N_{aPM} + \alpha * \sum_{k=1}^{m} N_{aCPU}^k \tag{13}$$

Using Equations 12 and 13, the ratio of consumed power in the cluster for each task scheduling pattern can be calculated as:

$$\frac{Pw_{active}(Cluster)}{Pw_{ful}(Cluster)} = \frac{Pw_0^z * N_{aPM} + \alpha * \sum_{k=1}^{m} N_{aCPU}^k}{Pw_0^z * N_{PM} + \alpha * \sum_{k=1}^{m} \mathrm{VMc_k}} \tag{14}$$

$$= \frac{Pw_0^z \left(N_{aPM} + \frac{\alpha}{Pw_0^z} * \sum_{k=1}^{m} N_{aCPU}^k\right)}{Pw_0^z \left(N_{PM} + \frac{\alpha}{Pw_0^z} * \sum_{k=1}^{m} \mathrm{VMc_k}\right)} \tag{15}$$

$$= \frac{\left(N_{aPM} + \mu \sum_{k=1}^{m} N_{aCPU}^k\right)}{\left(N_{PM} + \mu \sum_{k=1}^{m} \mathrm{VMc_k}\right)} = PowerC \tag{16}$$

where $\mu = \frac{\alpha}{Pw_0^z}$. For instance, for Altix XE320 with 16 number of CPUs, $Pw_{ful}(PM) = 115^{kw}$, and $Pw_0 = 40^{kw}$ [23], μ is equal to 0.1.

3.4 Length of VM Task Queues

If, in a possible best solution of the multi-objective task scheduling pattern, the number of tasks assigned to VM_k exceeds the number of its CPUs, extra tasks will be allocated to VM_k's task queue (see Fig. 2). In this model, we consider another objective function to optimize the task scheduling pattern by minimizing the length of VM task queues. This will reduce the task makespan and response time, as fewer tasks will be located in long queues and in waiting mode.

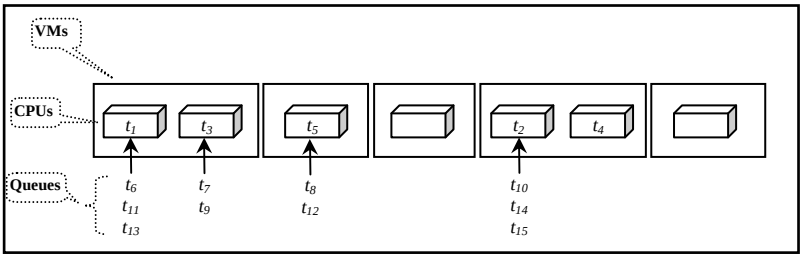

Figure 2. A sample for task scheduling pattern among VMs

The following objective is proposed to minimize the number of tasks in queues which would be created in any suggested possible optimal task scheduling pattern:

$$\gamma_k = \sum_{i=1}^{n} x_{ik} * \frac{DE_{ik}}{(RVM_m^k) * (RVM_c^k)} \tag{17}$$

where,

$$RVM_m^k = VMm_k - \left(\sum_{j=1}^{i} x_{jk} * tm_j\right) \tag{18}$$

$$RVM_c^k = VMc_k - \left(\sum_{j=1}^{i} x_{jk} * tc_j\right) \tag{19}$$

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

To calculate γ_k, the formula that was proposed in [10, 24] is improved by considering changes in the available capacity (CPU and memory) of the VMs after new tasks have been assigned to them. Equation 17 indicates that scheduling the task i to VM_k—which not only yields increasing DE_{ik}, but also reduces the amount of available CPU and memory of the corresponding VM— negatively affects VM_k's performance and execution time of its tasks. We have applied this formula to control the number of scheduled tasks to each VM and reduce the length of their task queue.

In Equation 17, $VMm_k \leq \left(\sum_{j=1}^{i} x_{jk} * tm_j\right)$ and/or $VMc_k \leq \left(\sum_{j=1}^{i} x_{jk} * tc_j\right)$ imply that there is no available memory or CPU in VM_k to execute extra tasks. In this situation, if RVM_m^k and/or RVM_c^k have a negative value, they will be multiplied to a small value (-10^{-3}), and if each of them is equal to zero, the value of 10^{-3} will be added to them. Since the values of RVM_m^k and RVM_c^k are multiplied by each other, this will increase the value of γ_k dramatically whenever one of them equals zero or has a nega-

tive value. Therefore, the probability of choosing this pattern as a possible optimal solution will be decreased. This prevents the assignation of tasks to VMs without available resources. In fact, coefficient γ_k controls the number of tasks assigned to VM_k to avoid creating a queue of tasks to be executed on VM_k, and maximizes its performance. According to this, for all iterations, after tasks have been assigned to VMs, the amount of memory of VMs that remains and the number of idle CPUs in each VM, are calculated as the available capacity of VMs to execute arrival tasks. The following formula is then used as an optimization objective in our proposed model to minimize VM task queue length and optimize the load balance:

$$\Gamma = \sum_{k=1}^{m} \gamma_k \tag{20}$$

3.5 The Multi-Objective Problem

The multi-objective task scheduling optimization model is described in this section based on predefined objective functions to minimize task transfer time, task execution cost, power consumption and task queue length, as follows:

Problem:

$$min\ f_{Time} = Ttrans \tag{21}$$
$$min\ f_{Cost} = Cexe \tag{22}$$
$$min\ f_{PowerCunsumption} = PowerC \tag{23}$$
$$min\ f_{TaskQueue} = \Gamma \tag{24}$$

Subject to

$$\sum_{k=1}^{m} x_{ik} = 1, \forall\ i = 1, \dots, n$$
$$x_{ik} \in \{0,1\}, \forall i = 1, .., n\ \&\ k = 1, \dots, m$$
$$0 \le r_p \le \tilde{C}_p\ , \forall p = 1,2, \dots, cp$$
$$N_{aPM} \le N_{PM}$$
$$N_{aPM}, N_{PM} \neq 0$$

4 Multi-Objective Task Scheduling Solution

The preliminary definition and explanation of MOPSO and MOGA methods are provided in this section. A MOPSO/MOGA-based algorithm for determining the optimal solution for our proposed model in Section 3 is also developed.

4.1 The Multi-Objective Particle Swarm Optimization Method

In majority of optimization problems, the objective functions are in conflict with each other and there is not unique solution for them. Therefore, the goal is to find

good trade-off solutions that represent the best possible compromises among the objectives [25]. A multi-objective optimization problem is defined as follows:

$$\text{Min} \quad \vec{\mathrm{F}}(\vec{\mathrm{x}}) = [\mathrm{f}_1(\vec{\mathrm{x}}), \mathrm{f}_2(\vec{\mathrm{x}}), \dots, \mathrm{f}_\mathrm{k}(\vec{\mathrm{x}})] \tag{25}$$

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) [26-29].

In the PSO method, particles are flown through hyper-dimensional search space. Changes to the position of the particles within the search space are based on the social–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 $\vec{X}_i(t)$ denote the position of particle i, at iteration t. The position of $\vec{X}_i(t)$ is changed by adding a velocity $\vec{V}_i(t+1)$ as follows:

$$\vec{\mathrm{X}}_\mathrm{i}(\mathrm{t}+1) = \vec{\mathrm{X}}_\mathrm{i}(\mathrm{t}) + \vec{\mathrm{V}}_\mathrm{i}(\mathrm{t}+1) \tag{26}$$

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

$$\vec{\mathrm{V}}_\mathrm{i}(\mathrm{t}+1) = \mathrm{W}\vec{\mathrm{V}}_\mathrm{i}(\mathrm{t}) + \mathrm{C}_1\mathrm{r}_1\left(\vec{\mathrm{x}}_{\mathrm{pbest}_\mathrm{i}} - \vec{\mathrm{X}}_\mathrm{i}(\mathrm{t})\right) + \mathrm{C}_2\mathrm{r}_2\left(\vec{\mathrm{x}}_{\mathrm{gbest}_\mathrm{i}} - \vec{\mathrm{X}}_\mathrm{i}(\mathrm{t})\right) \tag{27}$$

where C_1 is the cognitive learning factor and represents the attraction that a particle has towards its own success; C_2 is the social learning factor and represents the attraction that a particle has towards 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*; $\vec{\mathrm{x}}_{\mathrm{pbest}}$ is the position of the best particle of the entire swarm; and $\mathrm{r}_1, \mathrm{r}_2 \in [0,1]$ are random values [25]. In MOPSO all Pareto optimal solutions are stored in an archive and $\vec{\mathrm{x}}_{\mathrm{gbest}_\mathrm{i}}$ is chosen from this archive.

4.2 The Multi-Objective Genetic Algorithm

A multi-objective genetic algorithm (MOGA) is concerned with the minimization of multiple objective functions that are subject to a set of constraints. In this algorithm, an initial population whose scale is *N* is first randomly generated. The first generation child population is gained through non-dominated sorting and basic operations such as selection, crossover and mutation [30]. Then, from the second generation on, the parent population and the child population are merged and sorted based on fast non-dominated solutions. The crowding distance between individuals on each

non-dominated layer is calculated. According to the non-dominant relationship and crowding distance between individuals, appropriate individuals for forming a new parent population are selected. Lastly, a new child population is generated through basic operations of the genetic algorithm which iterates until the conditions of the process end can be met [31].

4.3 A MOPSO/MOGA-Based Algorithm

A MOPSO/MOGA-based algorithm is proposed to solve the proposed multi-objective task scheduling problem presented in Section 3. In the task scheduling model, there are n tasks $\{t_1, t_2, \dots, t_n\}$ that should be assigned to m VMs $\{vm_1, vm_2, \dots, vm_m\}$ to be executed (Table 2). MOPSO and MOGA methods are used to find the optimal task scheduling pattern, while minimizing task transfer time, task execution cost, power consumption and task queue length. All optimal suggested solutions (particle position/gene pattern) determined by MOPSO/MOGA techniques, are illustrated as $\vec{X}_{\mathrm{i}} = (x_1, x_2, \dots, x_n)$ vectors with continuous values, but their corresponding discrete values are needed to determine the ID of the VM chosen to execute tasks. Therefore, these continuous vectors $\vec{X}_{\mathrm{i}}$ are converted to discrete vectors $d(\vec{X}_{\mathrm{i}}) = (d_1, d_2, \dots, d_n)$ by applying the Small Position Value (SPV) rule [10].

Table 2. Task scheduling pattern.

Tasks	t_1	t_2	t_3	t_4	t_5	…	t_n
VM number = Particle position/gene pattern	vm_7	vm_4	vm_5	vm_7	vm_3	…	vm_m

Particle position (or gene pattern) in Table 2 are a possible solution, i.e., $d(\vec{X}_i) = (d_1, d_2, \dots, d_n) = (7, 4, 5, 7, 3, .., m)$, after converting the continuous values to discrete values. According to this, VMs: vm_7, vm_4, vm_5, vm_7, …, and vm_m are chosen to execute t_1, t_2, t_3, t_4, …, and t_n, respectively. Considering this fact, every particle/gene in our MOPSO/MOGA model has n dimensions to assign n tasks to m VMs. Every particle/gene will be assessed considering the predefined objective functions and all Pareto optimal solutions stored in an archive. In this paper, it is assumed that:

$$QoS\left(\vec{X}_i\right) = -\sum_{j=1}^{q} W_j f_j\left(\vec{X}_i\right), \{\forall\, \vec{X}_i \in \text{Archive}\} \tag{28}$$

where, q is the number of objective functions and W_j is the preference weight for every objective function $(f_j(\vec{X}_i))$. Pareto optimal solutions (archive members) are then ranked on the basis of the number of functions they minimize, and the maximum value of QoS. The top-ranking solution is chosen as the possible optimal solution $(\vec{X}_{gbest_i})$.

The first population in evolutionary algorithms is usually initialized randomly. In this paper, the first population is determined using VMs and task properties to accelerate the performance of MOPSO and MOGA. Evolutionary algorithms are therefore

expected to find the best solution faster because they start from the near best solution pattern. To achieve this, we select the VMs with a greater number of CPUs and a large amount of memory on active PMs as the new hosts for excess tasks. We apply the roulette wheel technique as it is applied in [2] to produce the first population for both MOPSO and MOGA. The Roulette Wheel selection method randomly selects a given choice from several options, based on the value of their winning probability. In this technique, the slots of a roulette wheel are first filled with the winning chance of the options, then the wheel is spun and an option is selected [2]. Equations 29 and 30 are proposed to determine the probability of choosing a task, and a host VM respectively. Then, based on the normalized winning chance, two roulette wheels are generated for tasks and VMs.

$$Chance\_VM_k = VM_{mips} + VMm_k + (\rho_1 * VMc_k) + (VMbw_k/\rho_2) \quad (29)$$

$$Chance\_Task_i = (DE_{ik}/\rho_3) + DT_{ik} \quad (30)$$

where, coefficients ρ_1, ρ_2, and ρ_3 have been determined based on the value of task and VM properties to make the same impact for all properties in determining winning chance. In this study, we have determined ρ_1= ρ_2=100 and ρ_3=1000 based on the data given in Tables 3 and 4. Fig. 3 shows a roulette wheel for VMs that is determined based on the information in Table 3.

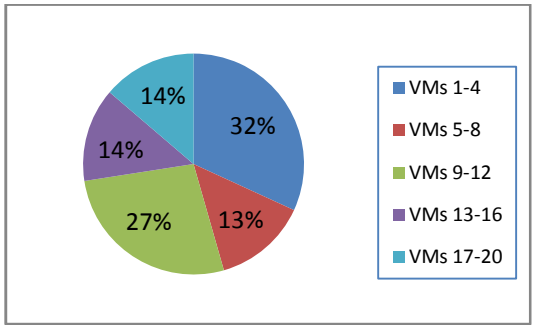

Figure 3. VM Roulette Wheel (Winning Probability)

After creating roulette wheels, a task and a VM are selected from the task and VM roulette wheels respectively as t_i and VM_j. Then the selected task (t_i) is deleted from task set, and a new task roulette wheel is generated for the new set of tasks. In addition, VM_j properties are updated by applying the following equations:

$$Available\_memory_{VM_j} = VMm_j - tm_i \quad (31)$$

$$Available\_CPU_{VM_j} = VMc_j - tc_i \quad (32)$$

If the value of $Available\_memory_{VM_j}$ or $Available\_CPU_{VM_j}$ is less than zero, this means VM_j has no available capacity to execute additional tasks. This VM is therefore deleted from the VM set and a new VM roulette wheel is generated for the new set of VMs based on their available capacity. After selecting all VMs as the new host for executing a set of tasks, and deleting those from the available set of VMs, all VMs are

applied again to execute the remaining tasks. These tasks are allocated to the VM task queues in waiting mode.

The corresponding steps for implementing the roulette wheel technique to initialize the first population for MOGA and MOPSO are described in Steps 2 of the MOPSO/MOGA-based algorithm, which is summarized as follows:

Step 1. Collect data and information about a possible set of host VMs and set of arrival tasks

Step 2. Initialize population: Determine new population (position and velocity of particles in MOPSO, or genes' pattern in MOGA) based on VM and task properties by applying roulette wheel technique as follows:

Step 2.1. Create the task and VM roulette wheels based on their properties by applying Equations 29 and 30

Step 2.2. $Count_{VM} = m$

Step 2.3. For i=1 to n

Step 2.3.1. Select a task (t_i) from task roulette wheel

Step 2.3.2. Select a VM (VM_j) from VM roulette wheel

Step 2.3.3. Delete t_i from the task set

Step 2.3.4. Create the task roulette wheel based on new task set and their properties by applying Equation 30.

Step 2.3.5. Calculate available VM memory and CPU by applying Equations 31 and 32

Step 2.3.6. If $Available\_memory_{VM_j} \leq 0$ or $Available\_CPU_{VM_j} \leq 0$ then

{

Delete VM_j from VM set,

$Count_{VM} = Count_{VM} - 1$,

If $Count_{VM} = 0$ then

{

Create new VM roulette wheel using original VM properties by applying Equation 29

$Count_{VM} = m$

}

}

else (if Step 2.3.6), create new VM roulette wheel based on available VM capacities by applying Equation 29

Next i

Step 3. Initialize an archive in which members are non-dominated solutions (n dimensions particles/genes whose position/pattern is a Pareto optimal solution)

Step 4. Convert continuous values of vector $\vec{X}_i$ to discrete vector $d(\vec{X}_i)$ using the SPV rule to determine the VM allocated for every arrival task.

Step 5. Determine the value of DE_{ij}, DT_{ij}, VMm_j, VMc_j, $Pcost_p$, $\mathrm{VM}_{\mathrm{mips}}^{\mathrm{k}}$, N_{aPM}, N_{aCPU}^{k}, RVM_m^k, RVM_c^k and $VMbw_j$ based on $d(\vec{X}_i)$ to calculate the value of every fitness function.

Step 6. Evaluate population according to defined fitness functions:

- Minimize task transfer time (Equation (21))
- Minimize task execution cost (Equation (22))
- Minimize power consumption (Equation (23))
- Minimize VM task queue length (Equation (24))

Step 7. Update the archive content by deleting dominated members from archive and store the Pareto optimal (non-dominated) solutions in the archive.

Step 8. Sort archive members based on the number of minimized functions and the maximum value of $QoS\left(\vec{X}\right)$

Step 9. Produce new population using MOPSO:

Step 9.1. Choose $\vec{X}_{gbest}$ from top sorted members in the archive

Step 9.2. Choose $\vec{X}_{pbest_i}$ for every particle: If the current position of the particle dominates best position of the particle, use current position as new best position for the particle

Step 9.3. Compute inertia weight and learning factors

Step 9.4. Compute new position of the particles and new velocity based on MOPSO formulations (Equations (26) and (27))

Step 10. Produce new population using MOGA:

Step 8.1. Reproduce best individuals using crossover and mutation

Step 11. If maximum iteration is satisfied, then

Step 11.1. Output $d(\vec{X}_{gbest})$ position for both MOPSO and MOGA as their best task scheduling patterns

Else

Step 11.2. Go to Step 3

5 Simulation Results

This section analyzes the efficiency of the proposed model. In Section 5.1 we first describe the simulation environment. Then, in Section 5.2 we explain how the Cloud-Sim package [17] is extended to implement the method, and lastly, the performance and evaluation is presented in Section 5.3.

5.1 Environment Description

We design the simulation environment by assuming that we have 15 PMs (data centers in CloudSim), 20 VMs, 3 cloud providers and 200 arrival tasks (cloudlets). Data and information about VMs and tasks (cloudlets) are summarized in Tables 3 and 4:

Table 3. Properties of VMs.

VM Id	MIPS	VM memory (Ram)	Bandwidth	The number of CPUs	VMM name
1-4	300	512	10000	4	Xen
5-8	200	256	1000	1	Xen
9-12	300	512	10000	2	Xen
13-16	200	256	1000	1	Xen
17-20	200	256	1000	1	Xen

Table 4. Properties of tasks.

Task Id	Length	File Size	Output Size	The number of required CPUs
1-20	250000	300	300	1
21-40	25000	200	300	1
41-60	250000	300	300	1
61-80	25000	200	300	1
81-100	250000	300	300	1
101-120	250000	300	300	1
121-140	25000	200	300	1
141-160	250000	300	300	1
161-180	250000	300	300	1
181-200	25000	200	300	1

5.2 Implementation

To implement the proposed method, we extend the Cloudsim toolkit [17] by using the MOPSO (MO-Jswarm package [32]) and MOGA (NSGA-II [33]) algorithms as the task scheduling optimization algorithms. The bindCloudletToVm() method in the DatacenterBrocker class of Cloudsim is responsible for allocating tasks to VMs according to the optimal task arrangement that results from the developed MOPSO/MOGA-based algorithms.

The objective functions in the proposed multi-objective task scheduling model are applied as the fitness functions in MOPSO and MOGA. In our model, we have 200 particles and the optimal results are obtained after the 2000th iteration of the MOPSO/MOGA algorithms.

5.3 Evaluation

To evaluate the proposed method, we first perform the simulation under the environment that we defined in Section 5.1. We evaluate our proposed multi-objective method of solving task scheduling problems with conflicting objectives by considering optimization time, cost, power consumption and workload from the following aspects:

- Compare the efficiency of the proposed four-objectives model with current bi-objective models in terms of optimizing cloud utilization, QoS and Job makespan.

- Compare the efficiency of MOPSO and MOGA in speed and reliability to find the highest value of QoS in different iterations.

To make the first comparison, we compare our method with the optimization methods proposed in previous works [10, 32, 34] in which just two aspects of QoS optimization were considered: (1) task transfer time, and (2) execution cost. The VM workload situation is also considered in this study to avoid multiple tasks being assigned to one VM, reducing its performance and increasing service response time. Our optimization model also has the objective of reducing the number of active PMs and busy CPUs to decrease power consumption and provider costs.

The graphs for task transfer time (f_{Time}), task execution cost (f_{Cost}), power consumption ratio ($f_{PowerConsumption}$), and VM task queue length ($f_{TaskQueue}$) obtained from the MOPSO/MOGA-based algorithm in 2000 iterations are illustrated in Figs. 4 and 5. The values of the axis on the right show the range of values of $f_{TaskQueue}$.

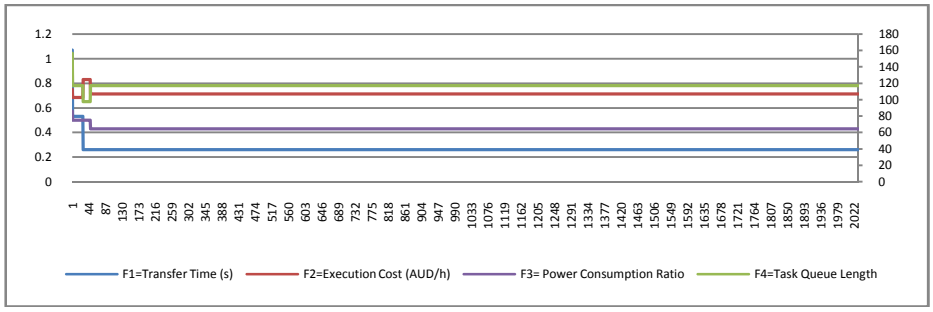

Figure 4. The value of objective functions: Task transfer time, task execution cost, VM task queue length, and power consumption ratio using MOPSO

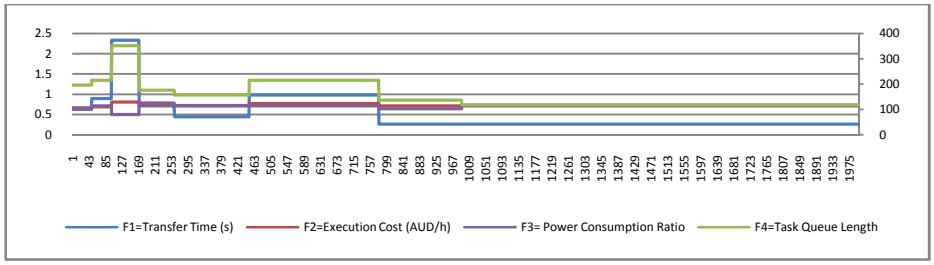

Figure 5. The value of objective functions: Task transfer time, task execution cost, VMs tasks queue length, and power consumption ratio using MOGA

To estimate the QoS that results from every method, Equation 33 is utilized by assuming the same preference weight ($w_1 = w_2 = w_3 = 1$) for time, cost and power consumption. We assign preference weight ($w_4 = 10^{-3}$) or ($w_4 = 1$) for a task queue determinator function based on its value. As discussed in Section 3.4, a small value -10^{-3} is multiplied by RVM_m^k or RVM_c^k to increase the value of γ_k when the required capacity by arrival tasks exceeds the amount of available capacity in the target VM. This prevents sending multiple tasks to a single VM. Therefore, in cases where a suggested

possible optimal task scheduling pattern assigns multiple tasks to certain VMs, the capacity required for executing the tasks assigned to those VMs would exceed the capacity of the VMs, then the value of $f_{TaskQueue}$ would be more than 10^3. In these situations $w_4=10^{-3}$ will be used in Equation 33, otherwise $w_4=1$. The value 10^{-3} for preference weight of $f_{TaskQueue}$ is determined as an aligner to make its value coherent with the other objective values and neutralize the effect of -10^{-3} coefficient in Equation 17. As the optimal values of these conflicting objectives are independent of one another, there is no need to normalize their weights in the QoS equation (Equation 28). However, as objective functions have different measurement units, we need to normalize their values to be able to calculate the QoS. To do this, we convert the range of objective functions to (0,1). For instance, the maximum value for the objective function f_{Time} produced by MOPSO and MOGA is 2300 seconds (see Figs. 4 and 5). Therefore, we convert f_{Time} ranging (0, 2300) to (0,1). As a result, the normalized value of f_{Time} i.e. Nf_{Time} for MOPSO is 0.11. The normalized value for all objective functions is calculated as Nf_{Time}, Nf_{Cost}, $Nf_{TaskQueue}$ and $Nf_{PowerConsumption}$ for all scenarios to determine the estimated value of QoS. In addition, considering the fact that we minimize the objective functions that have a negative impact on QoS, this method maximizes $QoS\ (\vec{X})$ as follows:

$$QoS\ (\vec{X}) = -[\mathrm{w}_1 * Nf_{Time} + \mathrm{w}_2 * Nf_{Cost} + \mathrm{w}_3 * Nf_{PowerConsumption} + \mathrm{w}_4 * Nf_{TaskQueue}] \quad (33)$$

MOPSO and MOGA are applied to optimize both the four-objective model and bi-objective model. In a bi-objective model, the optimal value of f_{Time} and f_{Cost} is determined by MOPSO and MOGA, then the corresponding value of $f_{TaskQueue}$ and $f_{PowerConsumption}$ is calculated by applying the optimal pattern of distribution tasks over VMs that results from these optimization algorithms. We run the algorithms 25 times for each comparison section, and the results are almost the same. The optimal results of all methods are summarized in Table 5.

Table 5. Comparison results.

Model	Algorithms	Transfer Time (s)	Execution Cost (AUD/h)	Power Consumption Ratio	Task Queue Length Coefficient	Estimated QoS	Max Execution Time (Makespan) (s)	Iteration	Number of Idle VMs
4 objectives	MOPSO	260	71	42%	116.00	-1.82	4400	49	45%
	MOGA	260	72	71%	117.18	-2.21	5100	992	40%
2 objectives	MOPSO	260	65	71%	156.2	-2.24	5400	148	40%
	MOGA	260	65	71%	156.2	-2.24	5400	709	40%

As can be seen from the comparison results in Table 5, the estimated QoS in the proposed model with four objectives achieves the highest QoS compared to bi-objective models that apply both MOPSO and MOGA. In addition, in the four objective models, the MOPSO determines the highest QoS in the lowest number of iterations (in 49th iteration) compared to MOGA, which determines its best possible solution in 992th iteration. MOPSO in bi-objective models is also faster than MOGA. As can be seen, the optimal load balancing between resources determined by our model also has the shortest makespan. This means the model has the ability to offer a task scheduling pattern with the lowest response time. In addition, the proposed model achieves lower power consumption and task queue length than the bi-objective models; however, the number of idle VMs in the cluster in all scenarios is almost the same.

Better load balancing leads to the highest level of power consumption because optimal load balancing requires a higher number of active PMs and CPUs. In this case, preference weights of the conflicting objectives $f_{TaskQueue}$ and $f_{PowerConsumption}$ can be changed according to the Service Level Agreement (SLA) to determine optimal task scheduling that satisfies the SLA criteria.

As a result, the proposed model that considers more aspects of task scheduling optimization, determines an optimal trade-off solution for the multi-objective task scheduling problem with objective functions that are in conflict with one another, and results the best possible compromise between objectives based on SLA, thereby increasing QoS.

6 Conclusion and Future Works

This study has developed a multi-objective model to optimize task scheduling which considers four aspects of the task scheduling optimization problem: task transfer time, task execution cost, power consumption, and task queue length (VMs' workload). We have also designed a MOPSO/MOGA-based algorithm to find the optimal solution for the proposed multi-objective task scheduling problem. To evaluate this method we extended the Cloudsim toolkit by applying MOPSO and MOGA as its task scheduling algorithms. The optimal solution determined by the MOPSO/MOGA-based algorithm is applied by the bindCloudletToVm() method in the DatacenterBrocker class of Cloudsim to assign tasks to VMs in an optimal way. The experimental results in the simulation environment show that the proposed optimization model has the ability to determine the best trade-off solutions compared to recent task scheduling approaches; it provides the best possible compromise between objectives and achieves the highest QoS. The experimental results also show that MOPSO is the most efficient and reliable algorithm since it not only determines the optimal task scheduling pattern with highest QoS, but also obtains the solution in the shortest possible time. The multi-objective model could be made part of the virtualization layer. This would enable data center operators to make use of this model for optimal load balancing. It would also give cloud providers the opportunity to boost their benefits by optimizing their preferred goals based on their determined objective weights in a

unique optimization model. They also have the ability to conduct sensitivity analysis on the value of optimized objective functions by changing their preferred weights. This sensitivity analysis process helps them to find the model with the highest level of benefit.

In future work, we will also consider task priorities and types in our optimization model, and cover more criteria of SLA. In addition, we will implement the proposed model in a real private cloud environment.

Acknowledgment

The work presented in this paper was supported by the Australian Research Council (ARC) under Discovery Project DP140101366. The Authors also would like to thank Mr Chaosong Nie for his kind help in implementing the MOGA algorithm.

References

[1] J. Taheri, A. Y. Zomaya, H. J. Siegel and Z. Tari. Pareto frontier for job execution and data transfer time in hybrid clouds. *Future Generation Computer Systems*, 2014, vol. 37, pp. 321-334.

[2] J. Taheri, A. Y. Zomaya, P. Bouvry and S. U. Khan. Hopfield neural network for simultaneous job scheduling and data replication in grids. *Future Generation Computer Systems*, 2013, vol. 29, pp. 1885-1900.

[3] E. Juhnke, T. D¨ornemann, D. B¨ock and B. Freisleben. Multi-objective scheduling of BPEL workflows in geographically distributed clouds. In *4th IEEE International Conference on Cloud Computing*, 2011, pp. 412-419.

[4] S. Tayal. Tasks Scheduling optimization for the Cloud Computing Systems. *International Journal of Advanced Engineering Sciences and Technologies*, 2011, vol. 5, no. 2, pp. 111-115.

[5] B. Song, M. M. Hassan and E. Huh. A novel heuristic-based task selection and allocation framework in dynamic collaborative cloud service platform. In *2nd IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2010, pp. 360-367.

[6] J. Li, M. Qiu, Z. Ming, G. Quan, X. Qin and Z. Gu. Online optimization for scheduling preemptable tasks on IaaS cloud systems. *Journal of Parallel and Distributed Computing*, 2012, vol. 72, no. 5, pp. 666-677.

[7] A. Salman, I. Ahmad and S. Al-Madani. Particle swarm optimization for task assignment problem. *Microprocessors and Microsystems*, 2002, vol. 26, no. 8, pp. 363-371.

[8] J. Li, J. Peng, X. Cao and H.-y. Li. A task scheduling algorithm based on improved ant colony optimization in cloud computing environment. *Energy Procedia*, 2011, vol. 13, pp. 6833-6840.

[9] Z. Lei, C. Yuehui, S. Runyuan, J. Shan and Y. Bo. A task scheduling algorithm based on PSO for grid computing. *International Journal of Computational Intelligence Research*, 2008, vol. 4, no. 1, pp. 37-43.

[10] L. Guo, S. Zhao, S. Shen and C. Jiang. Task Scheduling Optimization in Cloud Computing Based on Heuristic Algorithm. *Journal of Networks*, 2012, vol. 7, no. 3, pp. 547-553.

[11] W.-Y. Shieh and C.-C. Pong. Energy and transition-aware runtime task scheduling for multicore processors. *Journal of Parallel and Distributed Computing*, 2013, vol. 73, no. 9, pp. 1225-1238.
[12] X. Wang, Y. Wang and Y. Cui. A new multi-objective bi-level programming model for energy and locality aware multi-job scheduling in cloud computing. *Future Generation Computer Systems*, 2014, vol. 36, no. 0, pp. 91-101.
[13] N. B. Rizvandi, J. Taheri and A. Y. Zomaya. Some observations on optimal frequency selection in DVFS-based energy consumption minimization. *Journal of Parallel and Distributed Computing*, 2011, vol. 71, no. 8, pp. 1154-1164.
[14] A. Mahabadi, S. M. Zahedi and A. Khonsari. Reliable energy-aware application mapping and voltage–frequency island partitioning for GALS-based NoC. *Journal of Computer and System Sciences*, 2013, vol. 79, no. 4, pp. 457-474.
[15] S. Su, J. Li, Q. Huang, X. Huang, K. Shuang and J. Wang. Cost-efficient task scheduling for executing large programs in the cloud. *Parallel Computing*, 2013, vol. 39, no. 4–5, pp. 177-188.
[16] L. Wang et al. Energy-aware parallel task scheduling in a cluster. *Future Generation Computer Systems*, 2013, vol. 29, no. 7, pp. 1661-1670.
[17] R. N. Calheiros, R. Ranjan, C. A. F. De Rose and R. Buyya. Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services. *Arxiv preprint arXiv:0903.2525*, 2009.
[18] W. Cirne et al. Labs of the world, unite!!! *Journal of Grid Computing*, 2006, vol. 4, no. 3, pp. 225-246.
[19] A. Tchernykh, J. E. Pecero, A. Barrondo and E. Schaeffer. Adaptive energy efficient scheduling in Peer-to-Peer desktop grids. *Future Generation Computer Systems*, 2014, vol. 36, no. 0, pp. 209-220.
[20] B. Priya, E. S. Pilli and R. C. Joshi. A survey on energy and power consumption models for Greener Cloud. In *Advance Computing Conference (IACC), 2013 IEEE 3rd International*, 2013, IEEE, pp. 76-82.
[21] Y.-w. Zhang and R.-f. Guo. Power-aware scheduling algorithms for sporadic tasks in real-time systems. *Journal of Systems and Software*, 2013, vol. 86, no. 10, pp. 2611-2619.
[22] R. Buyya, A. Beloglazov and J. Abawajy. Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges. *arXiv preprint arXiv:1006.0308*, 2010.
[23] *Top 500 Supercomputing Sited*, [Online], Available: http://www.top500.org/system/176223.
[24] F. Ramezani, J. Lu and F. K. Hussain. Task-Based System Load Balancing in Cloud Computing Using Particle Swarm Optimization. *International Journal of Parallel Programming*, 2013, vol. 42, no. 5, pp. 739-754.
[25] M. J. Mahmoodabadi, A. Bagheri, N. Nariman-zadeh and A. Jamali. 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*, 2012, vol. 44, no. 10, pp. 1-20.
[26] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 2002, vol. 6, no. 2, pp. 182-197.

[27] M. J. Alves. *Using MOPSO to solve multiobjective bilevel linear problems*. Heidelberg: Springer, 2012.
[28] Y. Gao, G. Zhang, J. Lu and H.-M. Wee. Particle swarm optimization for bi-level pricing problems in supply chains. *Journal of Global Optimization*, 2011, vol. 51, no. 2, pp. 245-254.
[29] J. Lu, G. Zhang and D. Ruan. *Multi-objective group decision making: methods, software and applications with fuzzy set techniques*. London: Imperial College Press, 2007.
[30] N. Srinivas and K. Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 1994, vol. 2, no. 3, pp. 221-248.
[31] Y. Zhang, C. Lu, H. Zhang and J. Han. Active vibration isolation system integrated optimization based on multi-objective genetic algorithm. In *IEEE 2nd International Conference on Computing, Control and Industrial Engineering (CCIE)*, 2011, pp. 258-261.
[32] F. Ramezani, J. Lu and F. Hussain. Task scheduling optimization in cloud computing applying multi-objective particle swarm optimization. *International Conference on Service Oriented Computing (ICSOC)*, 2013, pp. 237-251.
[33] D. Hadka. *MOEA Framework A Free and Open Source Java Framework for Multiobjective Optimization*, [Online], Available: http://www.moeaframework.org/.
[34] H. Liu, A. Abraham, V. Snášel and S. McLoone. Swarm scheduling approaches for work-flow applications with security constraints in distributed data-intensive computing environments. *Information Sciences*, 2012, vol. 192, no. 0, pp. 228-243.