J Supercomput (2017) 73:3977–3998
DOI 10.1007/s11227-017-1995-9

Profile-based dynamic application assignment with a repairing genetic algorithm for greener data centers

Meera Vasudevan[1] · **Yu-Chu Tian**[1] · **Maolin Tang**[1] · **Erhan Kozan**[2] · **Weizhe Zhang**[3]

Published online: 6 March 2017

Abstract Data centers have become essential to modern society by catering to increasing number of Internet users and technologies. This results in significant challenges in terms of escalating energy consumption. Research on green initiatives that reduce energy consumption while maintaining performance levels is exigent for data centers. However, energy efficiency and resource utilization are conflicting in general. Thus, it is imperative to develop an application assignment strategy that maintains a trade-off between energy and quality of service. To address this problem, a profile-based dynamic energy management framework is presented in this paper for dynamic application assignment to virtual machines (VMs). It estimates application finishing times and addresses real-time issues in application resource provisioning. The framework implements a dynamic assignment strategy by a repairing genetic algorithm (RGA), which employs realistic profiles of applications, virtual machines and physical servers. The RGA is integrated into a three-layer energy management system incorporating VM placement to derive actual energy savings. Experiments are conducted to demonstrate the effectiveness of the dynamic approach to application management. The dynamic approach produces up to 48% better energy savings than existing application assignment approaches under investigated scenarios. It also performs better than the static application management approach with 10% higher resource utilization efficiency and lower degree of imbalance.

✉ Yu-Chu Tian
y.tian@qut.edu.au

[1] School of Electrical Engineering and Computer Science, Queensland University of Technology, GPO Box 2434, Brisbane, QLD 4001, Australia

[2] School of Mathematical Sciences, Queensland University of Technology, GPO Box 2434, Brisbane, QLD 4001, Australia

[3] School of Computer Science and Technology, Harbin Institute and Technology, Harbin 150001, Heilongjiang, People's Republic of China

Keywords Data center · Application assignment · Dynamic allocation · Energy efficiency · Genetic algorithm

1 Introduction

In today's information age, the rapidly growing online population realizes a subsequent explosion in data. As a result, the modern economy relies heavily on data centers for computing, network and storage services, to just name a few. Consequently, the energy required to power these systems is rising rapidly. The electricity consumed by data centers is predicted to rise from 7 to 12% of the global electricity consumption by 2017 [8]. A report by National Resources Defence Council (NRDC) has estimated that 91 billion KWh of electricity was consumed by the data centers in the USA in 2013. This statistic has a projected increase of 53% by year 2020. The NRDC further reports that there is a distinct gap in energy-efficient initiatives when comparing well-managed hyper-scale large data centers and the numerous less-efficient small- to medium-scale data centers. Typical hyper-scale large data centers are those from Microsoft, Google, Apple, Dell, Amazon and Facebook, which only share 5% of the global data center energy usage. Small- to medium-scale data centers are typically run by business companies, universities and government agencies [28]. This paper targets the latter class of widely deployed small- to medium-scale data centers.

Upto 40% of energy savings can be realized on deployment of energy-efficient measures [28]. According to the Greenpeace 2015 report [7], Apple, Google and Facebook lead the charge in deploying green solutions to reduce energy consumption and carbon footprint. Green solutions generally fall into three categories [12]: power infrastructure, cooling and IT solutions. Among these three categories of solutions, this paper limits itself to IT solutions. In general, a data center is managed through a three-layer architecture: application layer, virtual machine (VM) layer and physical machine (PM) layer. This is illustrated in Fig. 1. Server resource usage and ON/OFF operations are managed in the PM layer. The VM layer is responsible for VM management including VM placement and migration. The application layer assigns incoming applications to the previously generated VMs. For a data center, all those three layers work together to determine the overall energy consumption. Energy-efficient IT solutions such as virtualization, resource scheduling, server consolidation and application management

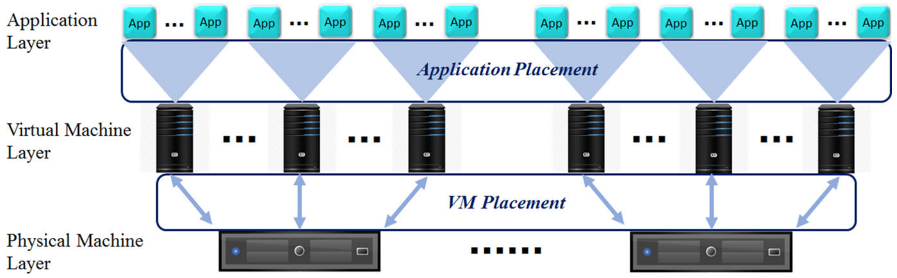

Fig. 1 Three-layer data center architecture

are deployed in one or more of these three layers. This paper demonstrates a new concept of utilizing profiles for energy-efficient dynamic application management.

The problem of dynamic assignment of applications to VMs in an energy-efficient way while satisfying performance levels is exigent. The challenges of the problem include: (1) How to quantitatively model a real-time service as a dynamic application management scheme using profiles; (2) how to maintain a trade-off between energy efficiency and performance with low overhead; and (3) how to develop a scalable, energy-aware and performance-efficient framework for real-time application assignment. Research and development with respect to large-scale distributed systems have been mostly driven by performance. However, the necessity for green and energy-efficient architectures and algorithms has become very real and emerging for reducing carbon footprint and the exorbitant energy costs. This becomes evident with the observation of the rapidly increasing number of computing and data-intensive applications. Such applications have mutable resource requirements based on user demands and submission times. Thus, it is difficult to minimize the energy consumption while preserving the quality of service (QoS) of the applications.

Our preliminary work in [23–25] has demonstrated theoretical energy savings using profiles for the application layer. Extending the previous work, the present paper deals with dynamic application assignment and also derives actual energy savings by considering a complete three-layer energy management system. Our profile-based dynamic application assignment to VMs is implemented together with a first-fit decreasing (FFD) VM placement policy. This will demonstrate actual energy savings at the server level. The main contributions of this paper include the following three aspects:

1. An energy-efficient dynamic application assignment framework is presented for real-time applications using profiles and implemented with a repairing genetic algorithm (RGA);
2. The finishing times of applications are estimated by using profiles to satisfy deadline constraints and reduce waiting times and assignment computation overhead; and
3. Strategies are developed to handle infrequent management scenarios such as new/random applications or failed/deactivated VMs.

The remainder of this paper is organized as follows. The notations used throughout this paper are listed in Table 1. Section 2 reviews related work and motivates the research. The dynamic research problem is described and formulated in Sect. 3. A profile-based dynamic application assignment framework for real-time applications is presented in Sect. 4. Section 5 gives a repairing genetic algorithm for an application assignment solution. The dynamic application assignment framework and RGA are evaluated in Sect. 6. Finally, Sect. 7 concludes the paper.

2 Related work

Data centers are the fundamental backbone of modern society. Subsequent to the explosive growth of connected users, networked devices, high-performance computing and data-intensive applications, the energy consumption of large-scale distributed

Table 1 Description of notations used in this paper

Notations	Description
α_j	Ratio of power consumed at max to min utilization of host V_j
A_i	Application, $i \in I = \{1, \ldots, n\}$
β_1, β_2	Coefficients in fitness function $F(X)$
C_{ij}	Energy cost of $A_i,\ i \in I$, assigned to $V_j,\ j \in J$
CPI	Cycles per instruction
$f_{\mathrm{vc}}^{\max}(j)$	Max CPU frequency (MHz) of $V_j,\ j \in J$
$f_{\mathrm{vc}}^{\mathrm{used}}(j)$	Used vCPUfrequency (MHz) of $V_j,\ j \in J$
$f_{\mathrm{c}}(k)$	Frequency of cores in $S_k,\ k \in K = \{1, \ldots, l\}$
$f_{\mathrm{c}}^{\mathrm{total}}(k)$	Total CPU frequency (MHz) of $S_k,\ k \in K$
$f_{\mathrm{c}}^{\mathrm{used}}(k)$ (%)	Total CPU usage (%) of $S_k,\ k \in K$
$F(\mathrm{obj}), F(X)$	Objective function and fitness function, respectively
i, j, k	Subscripts or indices for applications, VMs and PMs, respectively
I, J, K	Integer sets $I = \{1, \ldots, n\}$, $J = \{1, \ldots, m\}$, $K = \{1, \ldots, l\}$
$IC(i)$	Instruction count for $A_i,\ i \in I$, to execute successfully
$M_j^{\max}$	Memory capacity (MB) of $V_j,\ j \in J$
M_j^{used}	Used memory (KB) of $V_j,\ j \in J$
$M_r(i)$	Requested memory (KB) of $A_i,\ i \in I$
MIPS(j)	MIPS rate of $V_j,\ j \in J$
n, m, l	Total numbers of applications, VMs and PMs, respectively
$N_{\mathrm{c}}(k)$	Total number of cores in $S_k,\ k \in K$
$N_{\mathrm{r}}(i)$	Requested cores for $A_i,\ i \in I$
$N_{\mathrm{vc}}j$	Number of vCPUs of $V_j,\ j \in J$
P_k	Power consumed (W) of $S_k,\ k \in K$
$P_k^{\max}, P_k^{\min}$	Power at respect max and min utilizations of $S_k,\ k \in K$
S_k	Physical machine (PM) or server, $k \in K = \{1, \ldots, l\}$
$t_{\mathrm{c}}(j)$	Completion time of all applications on $V_j,\ j \in J$
$t_{\mathrm{d}}(i), t_{\mathrm{e}}(i), t_{\mathrm{f}}(i)$	Deadline, exec. time & finish time of $A_i,\ i \in I$, respectively
$t_{\mathrm{s}}(i), t_{\mathrm{w}}(i)$	Submission time and waiting time of $A_i,\ i \in I$, respectively
T_λ	Time slot ID, $\lambda \in \{1, 2, \ldots, \Lambda\}$
V_j	Virtual machine (VM), $j \in J = \{1, \ldots, m\}$
x_{ij}	Binary assignment decision variable
$X, \hat{X}$	Assignment decision matrix and its estimation, respectively
y_{jk}	Binary VM-server host constant
Δt	Time interval

systems has skyrocketed. This has motivated many research activities for development of energy-efficient measures for data centers.

Many existing application management solutions are deadline driven. They consider execution times [4,17,22] in order to increase resource computation efficiency. Addressing heterogeneous multi-cloud systems, Panda and Jana [18] have presented three task scheduling algorithms, which focus independently on completion time, median execution time and makespan, respectively. Ergu et al. [10] have proposed a rank-based task resource allocation model. The model weights tasks according to a reciprocal pairwise comparison matrix and the analytical hierarchy process. Yang et al. [29] have reported a host load prediction method for applications in cloud systems. The method uses phase space reconstruction to learn the best model of the evolutionary algorithm-based GMDH network. Song et al. [21] use online bin packing for resource provisioning while reducing the number of servers and meeting application demands. Li et al. [15] have developed Cress, a dynamic resource scheduling scheme for constrained applications. Cress attempts to maintain a trade-off between resource utilization and individual application performance. It employs a conversion method to dynamically adjust soft and hard constraints for fluctuating workloads. Fahim et al. [11] estimate the finishing times of the incoming tasks prior to allocation of the tasks to VMs. The allocation objectives include minimization of the degree of imbalance of VMs. In our work in the present paper, we will estimate the finishing times of applications using our model of profiles in order to satisfy deadline constraints while reducing energy consumption.

Advancements in high-performance computing (HPC) have led to inflation of system size and processing power. This contributes to high energy consumption. Mehrotra et al. [16] address this problem by using a utility-based control theoretical framework for scientific applications. Wang and Su [27] present a dynamic hierarchical task resource allocation scheme. In the scheme, the tasks and nodes are classified into different levels based on power and storage. The incoming tasks can only be hosted by the nodes on the same level. A smart green energy-efficient scheduling strategy (SGEESS) is developed by Lei et al. [14]. It considers renewable energy supply prediction and dynamic electricity price for real-time scheduling of incoming tasks. For sporadic real-time tasks, Zhang and Guo [30] present a static sporadic task low-power scheduling algorithm (SSTLPSA) and a dynamic version (DSTLPSA). Both algorithms have the objective of minimizing energy consumption. They use a combination of dynamic voltage scaling and power management to adjust task delay and processing speed while maintaining deadline constraints.

One of the most efficient techniques to maintain a trade-off between energy consumption and performance is the implementation of evolutionary algorithms for application management and resource provisioning. Wang et al. [26] propose a multi-objective bi-level programming model based on MapReduce for job scheduling. The model considers server energy-performance association and network to adjust job locality. Solutions are derived using a genetic algorithm enhanced with newly designed encoding/decoding and local search operation. Combining dynamic voltage frequency scaling, bin packing and genetic algorithm, Sharma and Reddy [20] have developed a hybrid energy-efficient approach for resource provisioning. Arroba et al. [1] present an automatic multi-objective particle swarm optimization method for dynamic cloud energy optimization. The method considers both power consumption and server temperature to derive an accurate server power model. The derived model can later be used to predict short-term power variations in data centers. ParaDIME, parallel distributed infrastructure for minimization of energy for data centers is reported by Rethinagiri et al. [19]. It includes power infrastructure and computing measures addressed at the software and hardware levels of a data center. Static energy profiles are used to identify those applications that can be run in a low energy mode. Subsequently, such applications are allocated to low-performance VMs and have limited parallelization. Our work presented in the present paper uses profiles in a new way together with implementation of a repairing genetic algorithm (RGA). It assigns applications to VMs such that a low energy consumption is incurred without compromising the efficiency of the system.

Extending our preliminary studies substantially, our work in the present paper supports dynamic application management. It uses the concept of profiles, which are introduced and established in our papers [23,24] for application assignment in data centers. In [25], we have further developed a profile-based static application assignment scheme with implementation through a simple penalty-based genetic algorithm. The genetic algorithm is chosen due to its ability of providing a feasible assignment solution on termination of the algorithm at any time [2]. This means the GA provides a feasible solution anytime when it is terminated due to time constraints or even an abnormal interruption to the algorithm. Our preliminary studies have considered synthetic profiles and derived theoretical energy savings by addressing only the application layer. In comparison, our work in the present paper derives profiles from actual workload logs and derives energy savings from a three-layer energy management system. However, due to the large data sets of the optimization problem, the performance of a simple steady-state GA deteriorates significantly with imperfect, slow or no convergence [31]. Therefore, a repairing genetic algorithm (RGA) is designed to solve the large-scale optimization problem. It enhances the GA by incorporating two components: (i) the longest cloudlet fastest processor (LCFP) and; (ii) an infeasible-solution repairing procedure (ISRP).

In the following sections, (1) a dynamic approach will be presented to deal with profile-based application assignment; (2) the research problem will be formally formulated by considering fluctuations in real-time application arrivals, resource demands and VM availability; (3) application finishing times will be estimated for satisfying deadline constraints and reducing waiting time and assignment overhead; (4) a real-time assignment strategy will be developed to address infrequent issues such as new/random applications or inoperative VMs; and (5) actual energy savings will be demonstrated through the three-layer energy management of data centers as shown in Fig. 1. The implemented VM placement policy is first-fit decreasing (FFD).

3 Problem formulation

Initially, the application, VM and server profiles have already been created off-line using data center workload logs. Then, the profiles are expanded and updated online as needed in real-time.

3.1 Characterizing application dynamics

In our application assignment problem, multiple real-time applications are to be allocated to VMs. The application assignment is required to satisfy deadline, waiting time and performance constraints. Consider real-time applications, $\{A_1, A_2, \ldots, A_i, \ldots, A_n\}$. The dynamic behavior of an application is characterized by varying arrival times, real-time constraints and resource demands. Thus, the real-time application $A_i, i \in I$, is characterized by the following parameters: (1) submission time $t_\mathrm{s}(i)$; (2) deadline constraint $t_\mathrm{d}(i)$; and (3) required resources such as cores $N_r(i)$, instruction count $IC(i)$ and memory $M_r(i)$. Therefore, the real-time application $A_i, i \in I$, is defined as

$$A_i = \langle t_\mathrm{s}(i), t_\mathrm{d}(i), N_r(i), IC(i), M_r(i) \rangle, \quad i \in I = \{1, 2, \ldots, n\} \tag{1}$$

The application profiles incorporate the above information. They also calculate waiting times, the maximum execution times, periodicity and finishing times in order to improve the application assignment efficiency. The waiting time $t_\mathrm{w}(i)$ is counted from the time instant $t_\mathrm{s}(i)$ when the application arrives till the time instant at which the application is allocated. Once the application is placed on a VM with allocated resources, it executes for a maximum execution time of $\max t_\mathrm{e}(i)$ for $i \in I$. The maximum execution time can be calculated by:

$$\max_{i \in I} t_\mathrm{e}(i) = t_\mathrm{d}(i) - t_\mathrm{s}(i), \quad i \in I \tag{2}$$

The length of the application $A_i, i \in I$, is represented by instruction count $IC(i)$, which is the number of instructions that need for A_i to execute successfully and is measured in million instructions. Periodicity is set to 1 if the application is regular to the data center or 0 if it is not periodic. The application releases the VM resources and exits the VM on execution completion.

The finishing time of the application $A_i, i \in I$, can be estimated using the application profiles before the allocation for satisfaction of the deadline constraints.

$$\text{Estimated } t_\mathrm{f}(i) = t_\mathrm{s}(i) + \frac{IC(i)}{\mathrm{MIPS}(j) \cdot N_r(i)} \tag{3}$$

Equation (3) is calculated for the requested number of cores ($N_r(i)$) to be fully utilized throughout the application runtime. In the event that the cores are not fully utilized or are shared with other applications, the estimated finishing time would be longer or shorted than the actual finishing time. However, with such an estimation, the application could be better assigned than the case without any information about the finishing time. Deviations of the estimation from the actual value could be handled dynamically through our dynamic assignment process.

3.2 Characterizing virtual machine dynamics

A virtualized data center consists of PMs $\{S_1, \ldots, S_k, \ldots, S_l\}$ and VMs $\{V_1, \ldots, V_j, \ldots, V_m\}$. From our collected information from a real medium-density data center, the processing capabilities of the PMs and VMs, such as MIPS rate, CPU frequency, memory and storage are pre-configured. They are reviewed every 6–12 months and updated if necessary. Therefore, the maximum server resources available to VMs are considered to be constant in this research.

For a PM S_k, the total CPU processing capacity $f_{\mathrm{c}}^{\mathrm{total}}(k)$ is the product of the frequency of the processors, $f_{\mathrm{c}}(k)$, and the number of cores, $N_{\mathrm{c}}(k)$. Similarly, for a VM V_j, the total CPU processing capacity $f_{\mathrm{vc}}^{\max}(j)$ is the frequency of the processors, $f_{\mathrm{c}}(k)$, times the number of vCPUs, $N_{\mathrm{vc}}(j)$. The million instructions per second (MIPS) rate, MIPS(j), of the VM is a ratio of $f_{\mathrm{vc}}^{\max}(j)$ to cycles per instruction, CPI.

$$f_{\mathrm{c}}^{\mathrm{total}}(k) = f_{\mathrm{c}}(k) \cdot N_{\mathrm{c}}(k) \tag{4}$$

$$f_{\mathrm{vc}}^{\max}(j) = f_{\mathrm{c}}(k) \cdot N_{\mathrm{vc}}(j) \tag{5}$$

$$\mathrm{MIPS}(j) = f_{\mathrm{vc}}^{\max}(j)/\mathrm{CPI} \tag{6}$$

For example, a server has 4 cores, each running at the frequency of 2 GHz. The server hosts 2 VMs with 1 vCPU each. In that case, the total CPU frequencies available in MHz from the server ($f_{\mathrm{c}}^{\mathrm{total}}(k)$) and VM ($f_{\mathrm{vc}}^{\max}(j)$) are 8000 and 2000 MHz, respectively. The processing rate of the VM is 1000 MIPS. If the CPU usage of the VM is above 80%, the VM is over-loaded and the VM status is set to 1. If the CPU usage falls below 20%, the VM is under-loaded and the VM status is set to 2. If there is a VM failure or deactivation, the VM status is set to 3. The dynamic VM model is described by the following parameters:

1. the status of the VM V_j; status(j)
2. the resource usage by all real-time applications: $f_{\mathrm{vc}}^{\mathrm{used}}(j)$, M_j^{used}, and
3. the linked list of all allocated applications: pointer.

3.3 Formulation of profile-based dynamic assignment

The decision matrix X at time t for application assignment to VM is given by:

$$X(t) = \left[x_{ij}(t)\right]_{n\times m}, \quad i \in I, j \in J \tag{7}$$

where

$$x_{ij}(t) = \begin{cases} 1, & \text{if } A_i \text{ is assigned to } V_j,\ i \in I, j \in J \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

At time t, the CPU and memory resources used by VM V_j is $f_{\mathrm{vc}}^{\mathrm{used}}(j, t)$ and memory $M_j^{\mathrm{used}}(t)$, respectively. The completion time of all applications on the VM is $t_{\mathrm{c}}(j)$. We have

$$f_{\mathrm{vc}}^{\mathrm{used}}(j, t) = \frac{\sum_{i=1}^{n} IC(i) \cdot x_{ij}(t)}{t_{\mathrm{c}}(j)} \cdot \mathrm{CPI} \quad (\mathrm{MHz}) \tag{9}$$

$$M_j^{\mathrm{used}}(t) = \sum_{i=1}^{n} M_r(i) \cdot x_{ij}(t) \quad (\mathrm{KB}) \tag{10}$$

In physical servers, CPU is the main power consumer compared with other system components like memory and storage, which have limited dynamic power ranges [3]. The power consumption of the physical server S_k at time t is calculated using the total CPU utilization percent $f_{\mathrm{c}}^{\mathrm{used}}(k,t)$ of the server in the corresponding time period:

$$f_{\mathrm{c}}^{\mathrm{used}}(k,t)[\%] = \frac{\sum_{j=1}^{m} f_{\mathrm{vc}}^{\mathrm{used}}(j,t)\cdot y_{jk}}{f_{\mathrm{c}}^{\mathrm{total}}(k)} \times 100 \tag{11}$$

$$P_k(t) = \left[\left(P_k^{\max} - P_k^{\mathrm{idle}}\right)\cdot f_{\mathrm{c}}^{\mathrm{used}}(k,t)/100\right] + P_k^{\mathrm{idle}} \tag{12}$$

where y_{jk} is 1 if VM V_j, $j \in J$, is hosted by server S_k, $k \in K$, and is 0 otherwise.

The ratio of power consumed at the maximum to minimum utilization of the host server is given by α_j. Therefore, the energy cost of allocating application A_i to VM V_j is calculated as a measure of CPU given by:

$$C_{ij} = \alpha_j \cdot IC(i)/\mathrm{MIPS}(j) \tag{13}$$

The constrained combinatorial optimization model for the assignment of a set of applications to VMs is given as:

$$F(\mathrm{obj}) = \min \sum_{j=1}^{M}\sum_{i=1}^{N} C_{ij}\cdot x_{ij} \tag{14}$$

$$\text{s.t.} \quad \sum_{i=1}^{n}\left[\frac{x_{ij}\cdot IC(i)}{\max\left(t_{\mathrm{e}}(i)\right)}\right] \le \mathrm{MIPS}(j), \quad \forall j \in J; \tag{15}$$

$$\sum_{i=1}^{N} x_{ij}\cdot M_r(i) \le M_j^{\max}, \quad \forall j \in J; \tag{16}$$

$$\sum_{j=1}^{M} x_{ij} = 1, \quad \forall i \in I; \tag{17}$$

$$x_{ij} = 0 \text{ or } 1, \quad \forall i \in I, j \in J. \tag{18}$$

The constraints in Eqs. (15) and (16) ensure that the allocated resources are within the total capacity of the VM. The constraint in Eq. (17) restricts an application from running on more than one VM. The binary constraint of the allocation decision variable x_{ij} is given by (18).

For a time interval of $\Delta t = 30$ min, the total daily energy consumption of all servers in the data center is given by:

$$\text{Total daily energy consumption} = \int_0^{48\cdot\Delta t}\left(\sum_{k=1}^{L} P_k(t)\right) d(t) \tag{19}$$

4 Profile-based dynamic application management framework

The workload of an application is dynamic in nature due to a number of factors such as change in resource requirements and load surges due to increased user requests. The work in [11] has reported two dynamic allocation algorithms: efficient response time load balancer and minimum processing time load balancer. While these algorithms update the allocation tables with respect to VM load, the update does not happen before the completion of processing current applications. In comparison, our dynamic assignment strategy presented in this paper updates the allocation tables periodically. Prior to the assignment of applications to VMs using RGA, the FFD algorithm sorts the PMs in decreasing order of resource capacity. Each active VM is placed onto the first server with adequate space remaining. All active VMs are eventually packed onto PM servers.

4.1 Dynamic application assignment

After application, VM and server profiles are built off-line, the real-time allocation of applications to VMs is initiated. The data center operation is divided into 30-min time slots Δt, the resulting time slot sequence is represented by $T_1, \ldots, T_\lambda, \ldots, T_\Lambda$. The periodic real-time allocation works as follows: The estimated task submission times in the application profiles are used to determine the arrivals of applications in specific time slots. In time slot T_λ, the estimated applications arrival in time slot $T_{\lambda+1}$ is batch processed for allocation of applications to potential VM hosts using the profiles and a repairing genetic algorithm (RGA), which will be presented later.

The potential assignment considers the following information:

1. the resource requirement history of the applications;
2. the load history of the VMs; and
3. the current assignment of applications to VMs

The resulting allocation solution is mapped onto an estimated assignment decision matrix $\hat{X}(T_\lambda)$. Therefore, during time slot $T_{\lambda+1}$, actual arriving applications are treated in a first in first out (FIFO) order. They are allocated to the predetermined host VMs according to the estimate decision matrix. After that, the actual decision matrix $X(T_\lambda)$ and profiles are updated.

Overhead of assigning profiled applications to VMs such that the application finishes execution within its deadline are calculated as below. It is normalized to the range of [0, 1], where 0 and 1 represent the lowest overhead and maximum overheads, respectively.

$$\text{Overhead} = \frac{t_\mathrm{w}(i)}{t_\mathrm{d}(i) - t_\mathrm{f}(i)} \tag{20}$$

where $t_\mathrm{f}(i)$ represents actual finishing time of the application. However, assignment overhead for applications finishing execution after their deadlines is set to a high value, e.g., 10. It is seen from Eq. (20) that the overhead of allocating profiled applications to VMs is small when the waiting time $t_\mathrm{w}(i)$ is low.

The process of the dynamic allocation scheme is described briefly in Algorithm 1. The algorithm executes for each time slot T_λ. It consists of two sequential processes: Process 1 in lines 1–5 for estimation of application allocation in the next time slot $T_{\lambda+1}$, and Process 2 from lines 7–17 for actual application allocation. Process 1 collects application profiles (line 2) and VM profiles (line 3). Then, it deploys RGA for an application assignment solution (line 4). After that, it creates an estimated allocation decision matrix $\hat{X}(T_{\lambda+1})$ in Line 5. In Process 2, applications arrive in a FIFO queue waiting for assignment to a VM (line 8). If an application is profiled and expected (line 9), then allocate the application to a VM using the estimated $\hat{X}_{\lambda+1}$ (line 10), and terminate the process. Otherwise, the application cannot be allocated to a VM from the estimated $\hat{X}_{\lambda+1}$. There are generally three scenarios: (1) If the application is not profiled, implying that it is a new application (line 12), then profile this new application (line 13); (2) the application is an unanticipated applications with random load; and (3) the application is a periodic existing application with different parameters. In all these three scenarios, the application is allocated to the first VM that meets the resource requirements (lines 14–16). After this allocation, the application and VM profiles need to be updated (line 17) before terminating the process.

Algorithm 1: Dynamic application assignment for each time slot T_λ

```
1  Process 1—Estimate allocation for $T_{\lambda+1}$
2  Collect application profiles for submission times within = $T_{\lambda+1}$;
3  Collect VM profiles, set VM status = 0, normal;
4  Deploy repairing genetic algorithm RGA (given later in Algorithm 2);
5  Create estimated allocation decision matrix $\hat{X}(T_{\lambda+1})$;
6
7  Process 2—Actual allocation for $T_\lambda$
8  Applications arrive in a FIFO queue;
9  if $A_i$ is profiled & expected then
10     Allocate $A_i$ to VM using $\hat{X}(T_{\lambda+1})$;
11 else
12     if $A_i$ is not profiled then
13         Profile this new application;
14     while $A_i$ is not yet allocated and a VM is available do
15         if VM resource availability $\geq$ Application requirement then
16             Allocate application $A_i$ to the VM;
17     Update application and VM profiles;
```

4.2 Dealing with infrequent applications

For infrequent applications described above in the three scenarios, dynamic application assignment to VM requires some special treatments. These treatments are described in the following.

Scenario 1: Arrival of new applications. Arriving applications that do not have profiles are new applications. For a new applications, its resource requirements need to be determined, and its profile will be created online. The new application is then allocated to the first available VM that meets the resource requirements and deadline constraints. The waiting times and allocation overhead of new applications are higher than those of expected applications with profiles. The overhead of allocating new applications falls between 0.75 and 1. However, there are not many new applications in the data centers of universities, government agencies and other small- to medium-scale business companies.

Scenario 2: Unanticipated existing applications with random load. Non-periodic applications with random load and variable submission/arrival times are considered as unanticipated applications. Although these applications have profiles, their resource demands are unknown prior to execution. Such applications are allocated promptly to the first available high-MIPS VM that satisfies the resource requirements and deadline constraints. The overhead for such allocations is in the range of 0.4–0.75. It is lower than that of allocation of new applications.

Scenario 3: Periodic existing applications with different parameters. This scenario considers a profiled periodic application that arrives at T_λ with different parameters. In this case, if the deadline of the application is less than the estimated finishing time on the predetermined VM, then the application is newly allocated to the first available high-MIPS VM that satisfies the resource requirements and deadline constraints. Otherwise, the application is allocated to the predetermined VM host without change. The overhead for such allocations falls between a wider range of 0.01–1 depending on the validity of the predetermined VM host.

Setting VM Status. Considering real-time usage of a VM, the status parameter of the VM profile is set to an integer from 0 to 3. The integer value 0 is for normal workload, 1 for over-loaded VM, 2 for under-loaded VM, and 3 for an inactive scenario due to VM failure or deactivation. If the status is normal (0), allocation proceeds successfully. The over-loaded and under-loaded CPU usage thresholds are set to 80 and 20%, respectively [6]. Once the thresholds are crossed, a repairing procedure discussed later in Sect. 5 is used to transfer one or more allocated applications from the affected VM to the next available VM. In the case of over-loading, the transfer of applications continue until the CPU usage falls below the higher threshold level. Under-loaded VMs are prioritised as available hosts for new and unanticipated applications with random load. If a VM becomes suddenly inactive due to failure or deactivation, then the executing applications stop and must be requested again by the user. These applications are then directed to the first available VM with high MIPS rate. Concurrently, the energy cost matrix is updated to reflect very high value for the failed VM. This ensures that the affected VM is not selected for hosting applications until the status returns to normal.

5 Repairing genetic algorithm

An integrated component of the dynamic application assignment framework described in Sect. 4 is the repairing genetic algorithm (RGA). It improves the general genetic algorithm (GA) with more heuristics. The motivation of choosing GA-based heuristics is due to GA's unique ability to deliver a feasible solution. For example, a typical scenario is that there is a short deadline to obtain an assignment solution. Another scenario is that the algorithm is interrupted abnormally. In either of these scenarios, GA guarantees a feasible solution at any time when GA is terminated. In addition, GA's parameters also enable control of the solution search space. This avoids becoming trapped in a local optima and thus drives the solution toward a global optima [2]. However, due to the large data set of the optimization problem, i.e., the large solution space, the performance of a simple steady-state GA degrades significantly with imperfect, slow or even no convergence [31]. Therefore, the simple GA is improved with the addition of two components: (i) the longest cloudlet fastest processor (LCFP) and (ii) an infeasible-solution repairing procedure (ISRP). This forms a repairing genetic algorithm (RGA).

RGA is implemented during Δt time intervals. At time T_λ, RGA uses profiles and the expected submission times of the applications to derive estimated allocations solutions for the next time slot $T_{\lambda+1}$, giving an estimated allocation decision matrix $\hat{X}$ (lines 5 and 6 in Algorithm 1).

In general genetic algorithms, there are two issues that affect the efficiency of deriving an application assignment solution: initial population and infeasible assignment. General steady-state GA methods utilize randomly generated initial population [25]. In contrast, this paper incorporates LCFP heuristics, which consider computational complexity of applications and computing power of processors, to generate an initial population. The advantages of using LCFP include faster convergence and better solutions. The steps involved in implementing LCFP are:

1. Sort applications $A_i, i \in I$ in descending order of execution time;
2. Sort VMs $V_j, j \in J$ in descending order of processing power (MIPS); and
3. Pack sorted applications into fastest processing VM.

For infeasible solutions, an ISRP process is employed to convert infeasible chromosomes to feasible solutions. An infeasible application assignment to a VM is denoted by a 'violation' indicator as a result of VM status violation or resource constraint violation:

$$\text{violation} = \begin{cases} 0, & \text{if resource constraints are not violated} \\ 1, & \text{otherwise} \end{cases} \tag{21}$$

When violation = 1, the applications assigned to infeasible VMs are re-assigned to other VMs until the violations are resolved. The ISRP pursues the following steps:

1. Identify VM $V_j, j \in J$, with violation = 1;
2. Calculate resource availability of next VM $V_{j'}, j' \in J$;
3. Assign the first application of V_j to new $V_{j'}$ if the new $V_{j'}$ has more than sufficient resources to host the application; and
4. Repeat above steps until violation = 0.

In RGA, the genetic operator settings are given in Table 2. The quality of the allocation solution is determined by the fitness function. A lower energy cost and higher resource utilization efficiency result in a higher fitness function. The fitness function is derived as:

$$F(X) = \beta_1 \cdot \overline{F}_{\text{obj}} - \beta_2 \cdot \frac{1}{m} \sum_{j=1}^{m} \left[\frac{f_{\text{vc}}^{\text{used}}(j)}{\text{CPI} \cdot \text{MIPS}(j)} + \frac{M_j^{\text{used}}}{M_j^{\max}} \right]. \tag{22}$$

Table 2 Genetic operator settings

Initial population	Randomly generated
Genetic encoding	Value encoding
Selection	Roulette wheel selection
Crossover	Uniform crossover with binary mask
Mutation	Select and exchange 2 offspring genes
Termination condition	Until max generations/suitable solution

The weights $[\beta_1, \beta_2]$ associated with the fitness function are set to [2, 1]. The multiplicative inverse of the objective function discussed in Eq. (14) is represented by $\overline{F}_{\text{obj}}$. In order to normalize and scale the objective function $\overline{F}_{\text{obj}}$ to a range of [1, 10], we use:

$$\overline{F}_{\text{obj}} = \frac{F_{\text{worst}} - F_{\text{obj}}}{F_{\text{worst}} - F^{\star}} \cdot \frac{F^{\star}}{F_{\text{obj}}} \cdot (\text{range}) + 1, \tag{23}$$

where the range (range) = 9. The best (minimized) and worst objective functions are represented by $F^{\star}$ and F_{worst}, respectively.

Now, we are ready to give a high-level description of RGA in Algorithm 2. Generally, RGA incorporates longest cloudlet fastest processor (LCFP) to generate initial population (lines 1 and 2). Then, it evaluates the fitness of each candidate chromosome (line 3). If the termination condition is met, output the best fit chromosome as solution. Otherwise, do the following operations sequentially for each generation (line 5):

Algorithm 2: Repairing Genetic Algorithm (RGA)

```
1  Find output of solutions generated by LCFP;
2  Initialize population with LCFP output;
3  Evaluate fitness of each candidate chromosome;
4  while Termination condition is not satisfied do
5      for Each Generation do
6          for Each chromosome do
7              Evaluate fitness;
8              if Chromosome is infeasible then
9                  Apply ISRP;
10                 Evaluate Fitness;
11             Parents selected using Roulette Wheel Selection;
12         for Parent chromosomes do
13             Apply uniform crossover as per crossover probability;
14             Mutate resulting offspring as per mutation probability;
15             Offspring chromosomes generated;
16         for Offspring chromosomes do
17             Evaluate fitness of new candidates;
18             Replace low-fitness chromosomes with better offspring;
19         for Each chromosome do
20             Select chromosomes for next generation;
21 Output the best fit chromosome as solution;
```

1. for each chromosome (line 6), evaluate the fitness (line 7); and if the chromosome is infeasible apply ISRP process and re-evaluate the fitness (lines 8–10), then select parents (line 11);
2. for parent chromosomes, do crossover, mutation and generation of offspring chromosomes (lines 12–15);
3. for offspring chromosomes, evaluate fitness of new candidates, and replace low-fitness chromosomes with better offspring (lines 16–18);
4. for each chromosome, select chromosomes for next generation.

After these sequential operations, the best fit chromosome is selected for output (line 21).

6 Case studies

This section undertakes experiments to demonstrate the efficiency of the profile-based dynamic application management framework with RGA. It begins with an introduction into experimental design. This is followed by a discussion of evaluation criteria. Then, experimental results and further discussions are presented.

6.1 Experimental design

In order to fully realize the efficiency of the dynamic application management framework, we have built profiles and conducted all experiments using workload traces from a real data center. This paper has used the MetaCentrum2 workload trace [13] provided by the Czech National Grid Infrastructure MetaCentrum. We have extracted seven days of workload from this trace. The total number of applications considered for our experiments varied per day as expected in a real data center. On average, there was 2583 applications per day, each with an instruction count of [5000, 10,000] million instructions and [500, 1000] KB of memory.

We have investigated a real data center with 100 physical servers, each with 2 to 4 CPU cores running at 1000, 1500 or 2000 MIPS and 8 GB of memory. The total number of VMs we considered is 300. Each VM has 1 vCPU, which we assume is equal to 1 server CPU core. The vCPU runs at 250, 500, 750 or 1000 MIPS and 128 MB of memory. The server power consumption values at maximum and idle utilizations (Eq. 12) are set at 350 and 150 W, respectively, as per our observation of the real data center.

The RGA has a population size of 200 individuals in each generation. The maximum number of generations is set to 200. Probabilities for crossover and mutation are set to 0.75 and 0.02, respectively. The mutation probability is set to a low value in order to control the exploration space. The termination condition is reached when there is no change in the average value and the maximum fitness values of strings for 10 generations.

6.2 Evaluation criteria

In order to evaluate the quality and efficiency of the solutions for profile-based dynamic application assignment to VMs, the dynamic scheme presented in this paper is compared with static-RGA application assignment and existing application assignment strategies: random (commonly used in data centers today) and first-fit [5]. The evaluation criteria include the following:

1. Energy efficiency: (a) energy consumption; and (b) statistical *T* Test analysis; and
2. Quality of solutions: (a) VM resource utilization; (b) makespan performance; (c) degree of imbalance; and (d) estimated finishing time performance.

6.3 Energy efficiency

The dynamic-RGA method presented in this paper is compared with three individual assignment methods used as benchmarks: static-RGA, random and first-fit [5]. In addition to application assignment, an FFD-based VM placement policy is implemented in our experiments with three-layer energy management as shown in Fig. 1. This enables calculation of actual energy savings and helps determine the efficiency of our dynamic application assignment solutions. Actual energy consumption of all servers across seven days is calculated using Eq. 19. The results are mapped onto Fig. 2. It is observed from this figure that both static- and dynamic-RGAs provide good results in terms of energy consumption. Considering the mean energy consumption of both methods over seven days, the energy savings of the dynamic-RGA is 7% more than static-RGA. When compared with existing application assignment strategies (benchmark), the dynamic-RGA is 48 and 34% more energy-efficient than the random and first-fit methods, respectively.

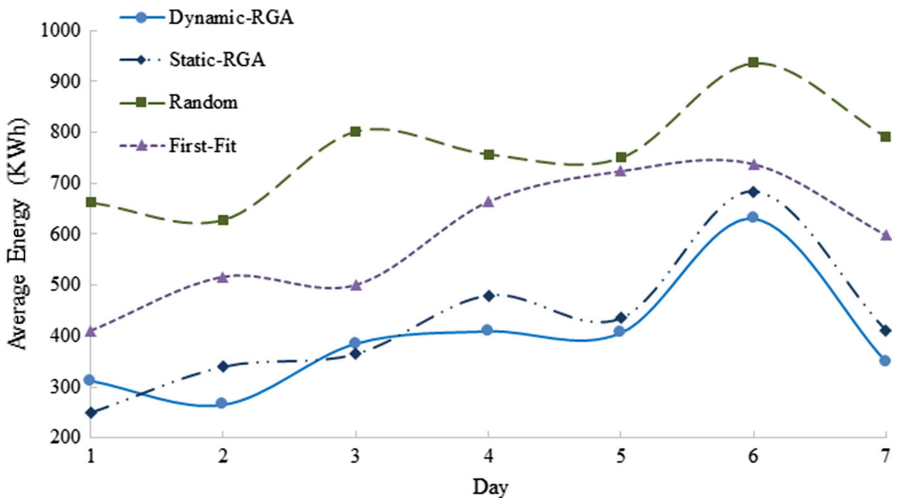

Fig. 2 Average energy consumption over 7 days for dynamic-RGA, static-RGA, random and first-fit strategies

Table 3 Daily energy consumption of the data center: static-RGA versus dynamic-RGA

Day	Energy consumption (KWh)		T value	SE	*df*
	Static-RGA	Dynamic-RGA			
1	250.57	316.53	−6.05	10.886	29
2	336.73	258.47	−4.96	15.764	29
3	370.63	384.60	−1.45	9.604	29
4	491.97	417.23	−4.75	15.714	29
5	441.77	391.43	−4.49	11.201	29
6	685.57	629.93	−10.58	5.257	29
7	423.63	343.23	−8.28	9.705	29

It is also observed in Fig. 2 that the dynamic-RGA behaves slightly worse than static-RGA on days 1 and 3. This is due to a large number of un-profiled applications on these two days. Our current dynamic assignment process allocates un-profiled applications to the first available VM rather than the most energy-efficient VM in order to meet deadline constraints. Once profiled, an application will be better allocated later when it appears again. Therefore, the dynamic-RGA will behave better and better as time goes on.

A paired t test is conducted to determine the confidence level of the experimental results for the two independent strategies of static-RGA and dynamic-RGA. As genetic algorithms are stochastic in nature, both strategies are individually run 30 times for each day. A two-tailed hypothesis is assumed, and the confidence interval is set to 95%. The null hypothesis is that there is no difference between the means of static-RGA and dynamic-RGA. Table 3 records the average energy consumption across seven days for the two strategies and the corresponding t-stat values. The two-tailed P value is less that 0.0001 and is extremely statistically significant. The results demonstrate that the difference between dynamic-RGA and static-RGA is significant, and thus, the null hypothesis is rejected.

6.4 Quality of solutions

The quality of solutions is determined by measuring resource utilization, makespan performance, VM degree of imbalance and estimated finishing time performance.

VM resource utilization The results of VM resource utilization on implementation of static-RGA and dynamic-RGA on the increasing number of applications are demonstrated in Fig. 3. Both static-RGA and dynamic-RGA realize a linear progression in terms of utilization efficiency. However, the dynamic-RGA performs upto 10% better in terms of resource utilization than static-RGA. The figure also demonstrates the scalability of the presented approach with respect to increasing problem size.

Makespan Makespan is the maximum completion time of all applications allocated to a VM:

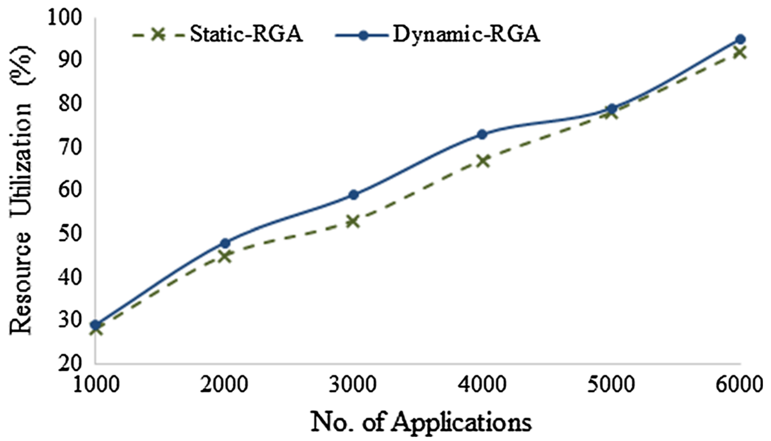

Fig. 3 VM resource utilization

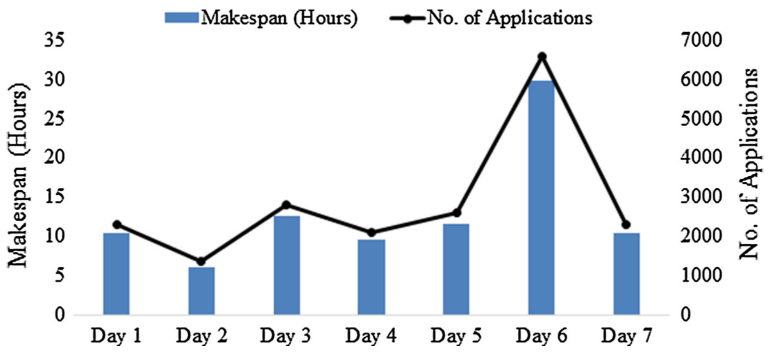

Fig. 4 Makespan of dynamic-RGA

$$\text{Makespan} = \max_{j \in J} t_c(j). \tag{24}$$

Completion time is the final time at which all applications conclude processing among all VMs. The number of VMs is a constant 300, and the number of applications varies every day. Figure 4 illustrates the maximum makespan incurred on implementing dynamic-RGA for the seven days under consideration. Makespan is linearly dependant on the number of applications.

Degree of imbalance (DI) The degree of imbalance represents the imbalanced distribution of load among VMs:

$$\text{DI} = \frac{\text{Makespan} - \min(t_c(j)}{\text{avg}(t_c(j)}. \tag{25}$$

The lower the DI, the more balanced the load distribution. DIs on implementation of the methods dynamic-RGA and static-RGA are 1.2 and 1.8, respectively. This demon-

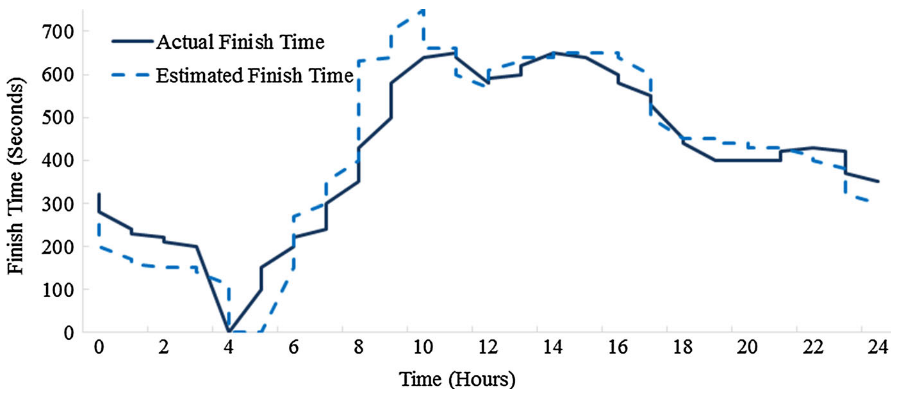

Fig. 5 Estimated finishing time performance

strates that dynamic-RGA is more efficient than static-RGA in terms of generating the least imbalanced load distribution.

Estimated finishing time performance The estimated and actual finishing times of the allocated applications over 24 h is shown in Fig. 5. The data shown in the figure are sampled at an interval of 1 h time slot. While the estimated finishing time deviates from the actual finishing time, the mean of the deviations over 24 h is as low as 5.551 s. Especially, all applications allocated through the estimated finishing times meet the deadline constraints, demonstrating the effectiveness of the dynamic application assignment approach presented in this paper.

6.5 Further discussions

Our experimental studies discussed above have used empirical data sets from a real data center. This makes the experimental scenarios more relevant to the real-world applications. However, changes in experimental parameters, coefficients and their ranges may impact the results. This is discussed from the following three aspects: multiple runs for each experimental scenario, impact on profiles and certain patterns that might be useful to simply the application assignment process.

Multiple runs for each experiment are discussed first. As the approach presented in this paper is non-deterministic, it is almost certain that every run of the algorithm will give a different result. In every run of the same experiment, the RGA progresses in different ways and consequently with different parameters or settings. From many runs of each experiment, an analysis can be carried out for statistical results, e.g., those shown in Table 3. Multiple runs for each of our experiments do not show an obvious difference in the results, implying the robustness of our algorithm.

Profiles of applications, VMs and PMs are not sensitive to environmental and parameter changes in our approach. They are built according to the requested resources and other features such as deadlines. As long as the requested resources and other requirements are the same for an application or a VM, the profile will be the same. For an incoming application that has not yet been profiled, assign it to the first available VM and at the same time profile it for future use. For an application that appeared before but with different settings or requirements, treat it as a new application.

In a real data center, many applications show some patterns. For example, there are some persistent or permanent applications, and also some periodic applications. There are some big application tasks and a huge number of tiny ones. Such patterns may help simply the matching process between applications and VMs, but have not been investigated in the present paper. Further research is being carried out from our group to address this issue [9] and the findings will be published later somewhere else.

7 Conclusion

The deployment of data center services on a massive scale comes with exorbitant energy costs and excessive carbon footprint. In this paper, a profile-based approach has been presented for energy-efficient dynamic application assignment to virtual machines without compromising the quality of service (QoS) such as resource utilization and workload balance. The assignment problem has been formulated as a constrained optimization problem, and a profile-based dynamic application assignment framework has been used to solve the problem. The framework is implemented with a repairing genetic algorithm (RGA). The finishing times of applications are estimated by using profiles to satisfy deadline constraints and reduce waiting times and assignment computation overhead. Strategies are developed to handle infrequent management scenarios such as new/random applications or failed/deactivated VMs. To derive actual energy savings, the dynamic assignment approach has been embedded into a three-layered energy management system. The VM management layer implements a first-fit decreasing (FFD) VM placement policy and the application management layer implements RGA. Experiments have been conducted to demonstrate the effectiveness and efficiency of the dynamic approach. For the investigated scenarios, the dynamic application assignment approach has shown up to 48% more energy savings than existing assignment approaches. Dynamic method also displays 10% more resource utilization efficiency and lower degree of VM load imbalance in comparison with the static application assignment method. Therefore, the profile-based dynamic assignment is a promising technique for energy-efficient application management in data centers.

Acknowledgements This work is supported in part by the Australian Research Council (ARC) under the Discovery Projects Scheme Grant No. DP170103305 to Y-C. Tian.

References

1. Arroba P, Risco-Martn JL, Zapater M, Moya JM, Ayala JL, Olcoz K (2014) Server power modeling for run-time energy optimization of cloud computing facilities. Energy Procedia 62:401–410
2. Bajpai P, Kumar M (2010) Genetic algorithm? An approach to solve global optimization problems. Indian J Comput Sci Eng 1(3):199–206
3. Barroso LA, Holzle U (2007) The case for energy-proportional computing. Computer 40(12):33–37
4. Calheiros RN, Buyya R (2014) Meeting deadlines of scientific workflows in public clouds with tasks replication. IEEE Trans Parallel Distrib Syst 25(7):1787–1796

5. Chandio AA, Xu CZ, Tziritas N, Bilal K, Khan SU (2013) A comparative study of job scheduling strategies in large-scale parallel computational systems. In: Proceedings of the 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications. IEEE, Melbourne, VIC, Australia, pp 949–957
6. Cisco (2013) Basic System Management Configuration Guide, Cisco IOS Release 15M&T, Chapter CPU Thresholding Notification, pp 1–7. Cisco Systems, Inc
7. Cook G, Pomerantz D (2015) Clicking clean: a guide to building the green internet. Technical report, Greenpeace
8. Corcoran PM, Andrae ASG (2013) Emerging trends in electricity consumption for consumer ICT. Technical report, National University of Ireland
9. Ding Z (2016) Profile-based virtual machine placement for energy optimization of data centres. Master's thesis, Queensland University of Technology, Brisbane, Queensland, Australia
10. Ergu D, Kou G, Peng Y, Shi Y, Shi Y (2013) The analytic hierarchy process: task scheduling and resource allocation in cloud computing environment. J Supercomput 64(3):835–848
11. Fahim Y, Ben Lahmar E, Labriji EH, Eddaoui A (2014) The load balancing based on the estimated finish time of tasks in cloud computing. In: Proceedings of the of the Second World Conference on Complex Systems (WCCS). Agadia, Morocco, pp 594–598
12. Huang R, Masanet E (2015) Chapter 20: Data Center IT Efficiency Measures
13. Klusacek D, Toth S, Podolnikova G (2015) Real-life experience with major reconfiguration of job scheduling system. In: Cirne W, Desai N (eds) Job Scheduling Strategies for Parallel Processing, pp 1–19
14. Lei H, Zhang T, Liu Y, Zha Y, Zhu X (2015) SGEESS: smart green energy-efficient scheduling strategy with dynamic electricity price for data center. J Syst Softw 108:23–38
15. Li Y, Han J, Zhou W (2014) Cress: dynamic scheduling for resource constrained jobs. In: Proceedings of 2014 IEEE 17th International Conference on Computational Science and Engineering (CSE), Chengdu, China, pp 1945–1952
16. Mehrotra R, Banicescu I, Srivastava S, Abdelwahed S (2015) A power-aware autonomic approach for performance management of scientific applications in a data center environment. In: Khan SU, Zomaya AY (eds) Handbook on Data Centers. Springer, New York, pp 163–189
17. Moens H, Handekyn K, De Turck F (2013) Cost-aware scheduling of deadline-constrained task workflows in public cloud environments. In: Proceedings of the 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM'2013), pp 68–75
18. Panda SK, Jana PK (2015) Efficient task scheduling algorithms for heterogeneous multi-cloud environment. J Supercomput 71(4):1505–1533
19. Rethinagiri SK, Palomar O, Sobe A, Yalcin G, Knauth T, Gil RT, Prieto P, Schneega M, Cristal A, Unsal O, Felber P, Fetzer C, Milojevic D (2015) ParaDIME: parallel distributed infrastructure for minimization of energy for data centers. Microprocess Microsyst 39(8):1174–1189
20. Sharma NK, Reddy GRM (2015) A novel energy efficient resource allocation using hybrid approach of genetic dvfs with bin packing. In: 2015 Fifth International Conference on Communication Systems and Network Technologies (CSNT 2015), Gwalior, India, pp 111–115
21. Song W, Xiao Z, Chen Q, Luo H (2014) Adaptive resource provisioning for the cloud using online bin packing. IEEE Trans Comput 63(11):2647–2660
22. Van den Bossche R, Vanmechelen K, Broeckhove J (2013) Online cost-efficient scheduling of deadline-constrained workloads on hybrid clouds. Future Gener Comput Syst 29(4):973–985
23. Vasudevan M, Tian Y-C, Tang M, Kozan E (2017) Profile-based application assignment for greener and more energy-efficient data centers. Future Gener Comput Syst 67:94–108
24. Vasudevan M, Tian Y-C, Tang M, Kozan E (2014) Profiling: an application assignment approach for green data centers. In: Proceedings of the IEEE 40th Annual Conference of the Industrial Electronics Society. IEEE, Dallas, TX, USA, pp 5400–5406
25. Vasudevan M, Tian Y-C, Tang M, Kozan E, Gao J (2015) Using genetic algorithm in profile-based assignment of applications to virtual machines for greener data centers. In: Proceedings of the 22nd International Conference on Neural Information Processing, Part II, Lecture Notes in Computer Science. Springer, Istanbul, Turkey, pp 182–189
26. Wang X, Wang Y, Cui Y (2014) A new multi-objective bi-level programming model for energy and locality aware multi-job scheduling in cloud computing. Future Gener Comput Syst 36:91–101
27. Wang Z, Xianxian S (2015) Dynamically hierarchical resource-allocation algorithm in cloud computing environment. J Supercomput 71(7):2748–2766

28. Whitney J, Delforge P (2014) Scaling up energy efficiency across the data center industry: evaluating key drivers and barriers (Issue Paper). Natural Resources Defense Council (NRDC)
29. Yang Q, Peng C, Zhao H, Yao Y, Zhou Y, Wang Z, Sidan D (2014) A new method based on PSR and EA-GMDH for host load prediction in cloud computing system. J Supercomput 68(3):1402–1417
30. Zhang Y-W, Guo R-F (2014) Power-aware fixed priority scheduling for sporadic tasks in hard real-time systems. J Syst Softw 90:128–137
31. Zhu K, Song H, Liu L, Gao J, Cheng G (2011) Hybrid genetic algorithm for cloud computing applications. In: Proceedings of the IEEE Asia-Pacific Services Computing Conference (APSCC). IEEE, Jeju Island, South Korea, pp 182–187