IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received January 28, 2021, accepted February 16, 2021, date of publication March 10, 2021, date of current version April 6, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3065170

Migration-Based Load Balance of Virtual Machine Servers in Cloud Computing by Load Prediction Using Genetic-Based Methods

LUNG-HSUAN HUNG, CHIH-HUNG WU, (Member, IEEE), CHIUNG-HUI TSAI, AND HSIANG-CHEH HUANG, (Senior Member, IEEE)
Department of Electrical Engineering, National University of Kaohsiung, Kaohsiung 811, Taiwan

Corresponding author: Chih-Hung Wu (johnw@nuk.edu.tw)

This work was supported in part by the Ministry of Science and Technology, Taiwan, under grant numbers MOST 108-2218-E-390-017 and MOST 108-2218-E-390-001 and Ministry of Education, Taiwan Higher Education SPROUT Project.

ABSTRACT This paper presents a two-stage genetic mechanism for the migration-based load balance of virtual machine hosts (VMHs) in cloud computing. Previous methods usually assume this issue as a job-assignment optimization problem and only consider the current VMHs' loads; however, without considering loads of VMHs after balancing, these methods can only gain limited effectiveness in real environments. In this study, two genetic-based methods are integrated and presented. First, performance models of virtual machines (VMs) are extracted from their creating parameters and corresponding performance measured in a cloud computing environment. The gene expression programming (GEP) is applied for generating symbolic regression models that describe the performance of VMs and are used for predicting loads of VMHs after load-balance. Secondly, with the VMH loads estimated by GEP, the genetic algorithm considers the current and the future loads of VMHs and decides an optimal VM-VMH assignment for migrating VMs and performing load-balance. The performance of the proposed methods is evaluated in a real cloud-computing environment, Jnet, wherein these methods are implemented as a centralized load balancing mechanism. The experimental results show that our method outperforms previous methods, such as heuristics and statistics regression.

INDEX TERMS Cloud computing, virtualization, load balancing, migration, genetic algorithm, gene expression programming.

I. INTRODUCTION

A. LOAD BALANCE IN CLOUD COMPUTING

Cloud computing is an Internet-based resource utility [1], [2]. Its central concept is ''everything can be a service''. In cloud computing, computing hardware and software resources are capsulized as web-services that can be accessed through the Internet. Three application types of cloud computing models are defined [3]: infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS). Among them, virtualization is a representative IaaS application, which provides computing infrastructure resources, such as computing power, data storage, networking, all in the form of web services [4], [5]. IaaS providers purchase and maintain physical computing and storage hardware and provide web services to users. With the virtualization technology, the users request IaaS providers for computation or storage resources as they own a ''virtual machine''(VM) without purchasing and maintaining physical hardware. The users can utilize VMs for deploying system/application software with a considerably lower cost of hardware procurement and possession.

An IaaS provider operates a server farm consisting of some computing and storage hardware, wherein some hosting servers (referring to as VMHs) provide virtualization services. A VMH may run one or many VMs, depending on the capacity of the VMH. If VMHs are not properly managed in a server farm, some VMHs may be busy running many VMs but some VMHs almost idle with few VMs. Managing loads of VMHs by adjusting the consumption of resources held by VMs for better cost-performance efficiency

The associate editor coordinating the review of this manuscript and approving it for publication was Christian Pilato.

while guaranteeing service level agreements (SLA) to the customers is an essential issue in IaaS [6]–[9].

Migration of VMs among VMHs is one of the methods for load balance of VMHs, which is to move the workload of VMs from one overloaded VMH to other VMHs, trying to make the workload of all VMHs evenly. Three phases of work are involved in this process: detection, decision, and action. The detection phase is to detect if an imbalance occurs in a server farm. The decision phase is to decide if the migration of VMs is needed and select the VMs for migration and VMHs for accepting these VMs. The action phase is to suspend the VMs selected, migrate them from one VMH to others, and restart them after the migration. During the migration, only the VMs to be moved are suspended and restarted, and other VMs in the VMHs are still running. Thus, the workload of VMHs is not static; migration of VMs may cause the workload of the target VMHs to much worse. Therefore, an effective load balancing mechanism is essential yet difficult for the management of VMHs.

Load-balance is a typical but essential research topic in parallel or distributed systems, and many machine learning-based methods have been proposed, such as [10]–[12]. The following discusses some recent methods for load balancing in cloud computing. Alonso-Calvo *et al.* presented in [13] an application-level load balancing system for applications running on VMs. Doong *et al.* presented in [14] a multi-kernel support vector regression model for modeling VMs performance. Hu *et al.* developed a scheduling strategy for load balancing of VM resources using GA that refers to historical data and the current state of the system [15]. A capacity allocation algorithm is presented in [16] that coordinates multiple distributed resource controllers executing in geographically distributed cloud sites. Pang *et al.* presented in [17] a hybrid method that employs the estimation of distribution algorithm to estimate possible solutions of VM loads and then uses GA to adjust these solutions. HEELS [18] is a heuristic task deployment approach based on clustering and Glowworm Swarm Optimization and used for long-term load balancing of a cloud framework with edge computing. A hybrid metaheuristic is proposed in [19] that hybridizing artificial bee colony and ant colony optimization for load balancing of VMs in the cloud.

These methods consider the load balance problem as job-assignment optimization and mainly focus on developing optimization algorithms for fast convergence. However, they assume the load of VM/VMH is static and do not consider the cost of migration and the load of VMHs after balancing, resulting in limited effectiveness in real environments.

B. MOTIVATION

Based on the above discussions, an effective and efficient load balancing mechanism must consider the following. A load balancing mechanism works in a dynamic environment consisting of many running VMs and VMHs. It needs to monitor, detect, and decide how to release the imbalance situation of VMHs. An effective load balancing mechanism can take actions that release an overloaded VMH and do not result in overloads on another. Also, efficiency and low overhead are required for a balancing mechanism. Simple balancing methods like fixed or heuristic-based rules work efficiently but not effectively since they may derive overloading of other VMHs. Advanced balancing methods like statistics-based can work effectively; however, they need to calculate many sophisticated operational parameters and require considerable computation power to decide migration targets.

Because there are diverse IaaS implementations, some assumptions are adopted in this study.

1) A VM operates on only one VMH at a time, and multiple VMHs share a data storage. The amount of resource consumption of a VM cannot exceed the host VMH can offer.
2) There are many types of load in computer systems, such as CPU/GPU computing load, memory/disk storage, network transmission. This study focuses on the CPU load of VMHs, which is mainly discussed in the literature.
3) A centralized load balancing mechanism is employed for monitoring and managing all VMHs. The mechanism detects the number, locations, and all VMH workload parameters periodically.

This study was motivated by the authors' daily experiences in managing VMHs. Two genetic-based methods are developed and integrated for load balance. First, performance models of virtual machines (VMs) are extracted from their creating parameters and corresponding performance measured in a cloud computing environment. The gene expression programming (GEP) is applied for generating symbolic regression models that describe the performance of VMs and are used for predicting loads of VMHs after load-balance. Secondly, with the VMH loads estimated by GEP, the genetic algorithm considers the current and the future loads of VMHs and decides an optimal combination of VM-VMH assignments for migrating VMs and performing load-balance. Our proposed methods work effectively and efficiently for balancing VMH workloads in a dynamically working VMH environment. The performance of the proposed method is evaluated in a real cloud-computing environment. The experimental results show that our method outperforms previous methods.

As stated in the literature, load balance is a critical issue for managing VMH servers in cloud computing. Using GEP, the idea of load prediction of VM for optimizing VM-VMH assignment helps improve the stability of the load balance mechanism. The white-boxed GEP expression of VM load behaviors can be easily integrated with the balance mechanism. GA provides an intuitive and fast method of deciding VM-VMH assignments for load balance and flexibly works with various objective functions for different management purposes. The genetic-based methods, GA and GEP, can be easily implemented by the cloud administrator. The performance and usability of the proposed methods are evaluated and proved to be valid in a real cloud environment.

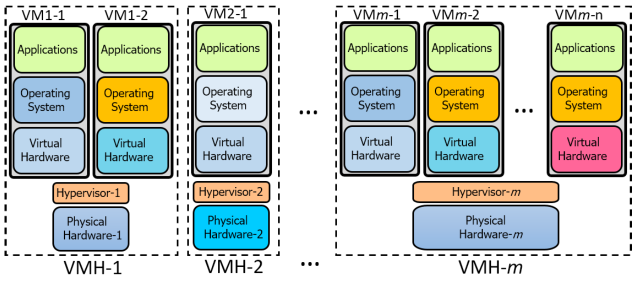


FIGURE 1. VMs, hypervisor, and a server farm with *m* VMHs.

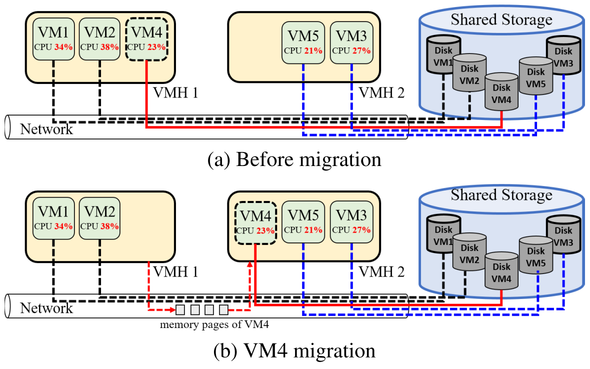


FIGURE 2. Live migration with shared storage.

II. BACKGROUNDS

A. VIRTUALIZATION

The virtualization technology is to establish a software layer called a hypervisor on a physical hardware platform. A hypervisor is an operating system that protects the resources of a VMH and manages the requests and responses from VMs. Depending on the application environments, type-1 hypervisors directly running on VMH's hardware and type-2 running on the VMH's OS are defined. For more details about hypervisors, please refer to [20]. A VM is an abstract machine defined by virtual processors (vCPUs), virtual memory (vRAM), and virtual hard drives (vHDs) provided by a VMH. A user program running on a VM cannot directly access the physical hardware. Instead, all resources are wrapped by the VMH's hypervisor. Generally, multiple VMs are executed on and managed by a VMH. Fig. 1 presents a collection of VMHs (referring to as a VMH server farm), where each VMH serves for one or many VMs. Commonly used hypervisors include VMware vSphere [21], KVM(kernel-based virtual machine) [22], and Microsoft Hyper-V [23].

Conceptually, a VM is composed of a "configuration file" describing the specifications of vCPU, vRAM, and vHD and a "disk image file" retaining the user data (vHD). A volatile "memory page" is generated in the VMH's main memory when the VM is running. When a VM is created, the above files are created on the VMH's storage. Therefore, deleting, copying, or moving a VM means deleting, copying, or moving these files. One of the advantages of virtualization is the easy migration of VMs, which means that one VM can move from one VMH to another VMH. With migration, VM can execute on different VMH platforms. During the migration, if the VM is in execution, the VM needs to be suspended first. In addition to moving the VM files (configuration and disk image), the memory pages associated with the VM must also be moved from the current VMH's main memory to the target VMH. After all the movements are completed, the VM can be activated again.

During migration, a VM may be suspended for seconds, even dozens of minutes, depending on the sizes and storage methods of vHD and vRAM. Additionally, migrating a VM incurs the transfer of vRAM pages over the network. The larger the vRAM, the more considerable the amount of data to be transferred, and the greater the system's burden. The so-called "live migration" [24] is a fast re-configuration of vHD and vRAM for migration with a shorter suspending time with which the VM users may not even notice the suspension in the migration process. Live-migration is commonly implemented with shared storage. VM migration can be easily and quickly done by re-assigning the ownerships and disk mapping files instead of physically transferring vHDs (huge sizes). Fig. 2 illustrates the scenario of live migration with shared storage between two VMHs. Suppose the administrator selects VM4 to migrate. The disk mapping files associated with VM4 are reconfigured in the storage, and the memory pages are transferred from VMH1 to VMH2. With live migration, load balance can be more efficient and reduce the impact on working VMs [25], [26].

B. PROBLEM FORMULATION

The load of a VM is dominated by factors like CPU usage, memory size, network speed, etc. Assume that v is a VM, and $L_V(v)$ represents its load. The $L_V(v)$ can be described as a function, as shown in Eq. (1).

$$L_V(v) = f(u_1, u_2, \ldots, u_k), \tag{1}$$

where $u_1, u_2, \ldots, u_k$ are the aforementioned k factors that dominate L_V. A VMH may serve many VMs. The workload of a VMH is the combination of the load from all its VMs and its operating system. Assume h is a VMH with n VMs $(v_i, v_2, \ldots, v_n)$ running on h. The load $L_H(h)$ of h can be described as follows.

$$L_H(h) = L_0(h) + \sum_{i=1}^{n} L_V(v_i), \tag{2}$$

where, $L_0(h)$ is the system load of h and $L_V(v_i)$, $1 \leq i \leq n$, is the load of the VM v_i. Usually, each VM's load is protected by SLA and should be above a reasonable level to maintain good service quality. Also, by the quota contract, VM's load is limited. Eq. (3) describes the upper and lower bounds of $L_V(v_i)$.

$$l_*(v_i) \leq L_V(v_i) \leq l^*(v_i). \tag{3}$$

Usually, $l_*(v_i)$ and $l^*(v_i)$ are constants set by the administrator of VMHs according to the SLA and contract requirements. If $L_V(v_i) \geq l^*(v_i)$, the v_i's load is too high, and it may not perform the user's tasks smoothly. If $L_V(v_i) \leq l_*(v_i)$, the load is low, causing a waste of resources of the VMH.

Suppose there are p VMHs, $h_1, \ldots, h_p$, operating in a server farm. The overall load of each VMH must also be within a reasonable range and can be described as Eq. (4).

$$L_*(h_j) \leq L_H(h_j) \leq L^*(h_j), \tag{4}$$

where $1 \leq j \leq p$. Also, $L_*(h_j)$ and $L^*(h_j)$ are control parameters set by the administrator. If $L_H(h_j) \geq L^*(h_j)$, the VMH is overloaded, and the VMs running on h_j may not obtain sufficient resources, degrading the performance and even service quality. If $L_H(h_j) \leq L_*(h_j)$, the computing resources are idle and wasted.

With the above equations, to keep the load of a set of VMHs balanced is to keep all $L_H(\cdot)$s evenly, i.e.,

$$L_H(h_1) \approx L_H(h_2) \approx \cdots \approx L_H(h_p), \tag{5}$$

and to preserve the consistency of Eq. (4) for all VMHs. If Eq. (5) is corrupted due to the overload of some VMHs, some VMs associated with these VMHs may need to be turned off or migrated to other VMHs for load-balancing. The load of all VMHs is expected to be balanced after the migration of VMs. Suppose h_s is a VMH about to become overloaded and h_t is a VMH in safe load, i.e., $L_H(h_s) > L^*(h_s)$ and $L_H(h_t) \ll L^*(h_t)$. If migration of q VMs, $v_1, \ldots, v_q$, from h_s to h_t is performed, the load of h_s and h_t changes as follows.

$$L'_H(h_s) = L_H(h_s) - \sum_{i=1}^{q} L_V(v_i), \tag{6}$$

$$L'_H(h_t) = L_H(h_t) + \sum_{i=1}^{q} L_V(v_i) \tag{7}$$

Therefore, load balancing by the migration of VMs can be considered as a combinatorial problem, wherein the best set of VMs are selected from VMHs and migrated to proper hosts and Eq. (4)–Eq. (5) are always observed either before or after the VM migration. The following issues are considered in this study.

- The load of VMHs is expected to remain balanced after the migration of VMs. It is necessary to predict the load of VMHs to prevent the target hosts from becoming overloaded after the migration. One method to obtain the descriptive expression of $L_V(v_i)$ in Eq. (2) is to collect a sufficient amount of data and find a statistics or regression model accordingly. There are two types of learning methods, black-box and white-box [27]. The main difference between them is the explainability and readability of the regression model obtained. An explainable VM load model can be integrated and adjusted more easily with the management system from the system administration perspective.
- Choosing the best set of VMs and target hosts for migration is a job-assignment optimization problem, usually a time-consuming combinatorial explosive problem. An efficient and effective selection mechanism is needed for load balance and painless migration.

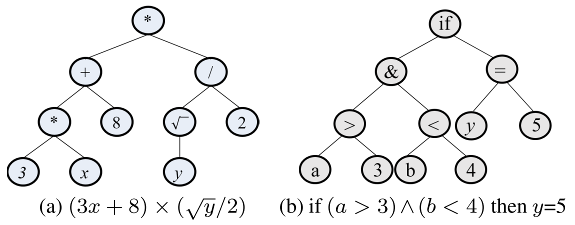

*
+
/
*
8
√
2
3
x
y
(a) $(3x+8) \times (\sqrt{y}/2)$
if
&
=
>
<
y
5
a
3
b
4
(b) if $(a > 3) \wedge (b < 4)$ then y=5

FIGURE 3. GP trees.

III. GENETIC METHODS

There are two goals in this study. The first one is to describe the load of VMs as a white-box model. The second one is to decide on the best VM-VMH assignment for migration. This research adopts two genetic methods, genetic algorithm and genetic expression programming, for achieving these goals.

A. GENETIC ALGORITHM

The genetic algorithm (GA) [28] is a stochastic search optimization method that mimics natural genetic systems' mechanics. With GA, solutions to domain problems are encoded as chromosomes that are iteratively processed by genetic operators, reproduction, mutation, crossover, etc. The ones with better fitness are retained. GA is well known for its efficient exploitation of better solutions via search in the vicinity of known ones. GA has many advantages over traditional search methods: (1) it provides a simple and direct representation of solutions; (2) it searches solutions with a population of potential ones; and (3) it works with probabilistic transition rules instead of deterministic ones. GA has been successfully applied to many combinatorial optimization problems and is widely used in many applications. Genetic programming (GP) [29], is a genetic method for evolving programs or mathematic formulas. GP uses a tree structure to represent a program or formula, as shown in Fig. 3. The genetic operators in GP are similar to those in GA (mutation, crossover, and reproduction); however, they are applied as node-renaming, pruning, and recombination of subtrees. Because tree structures are complicated and computationally inefficient for implementation, some improvements are proposed, such as the following.

GA is a type of metaheuristic algorithms that applies specific search schemas for solving complex optimization problems. Each metaheuristic algorithm employs a unique representation for problem description and a specialized mechanism for exploring the search space and is useful for particular applications. GA is adopted in this study due to the following reasons.

- GA provides a direct and intuitive representation for describing the VM-VMH assignment and is easy to implement.
- Many metaheuristic algorithms are working on an established objective function (fitness function), which may be changed for various load balance considerations (see Section VI).
- For various optimization subjects, only the objective function's calculation needs to be changed without

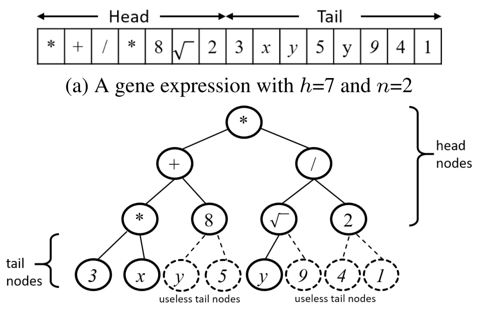

Head
Tail
* + / * 8 √ 2 3 x y 5 y 9 4 1
(a) A gene expression with h=7 and n=2
head nodes
tail nodes
useless tail nodes
useless tail nodes
(b) Binary tree expression of (a)

FIGURE 4. A GEP example–gene and tree expressions.

TABLE 1. Operators used in GEP.

Name	Definition	Name	Definition
Addition	$(x+y)$	Inverse	$1/x$
Subtraction	$(x-y)$	Power of 2	x^2
Multiplication	$(x*y)$	Negation	$-x$
Division	(x/y)	Average of 2 inputs	$avg(x,y)$
No operation	x	Max. of 2 inputs	$max(x,y)$
Complement	$(1-x)$	Min. of 2 inputs	$min(x,y)$
10^x	$pow(10,x)$	Exponential	$exp(x)$
Floor	$floor(x)$	Absolute value	$abs(x)$
Ceiling	$ceil(x)$	Natural logarithm	$ln(x)$
Square root	$sqrt(x)$	10-based Logarithm	$log(x)$
Power	$pow(x,y)$	Floating-point remainder	$mod(x,y)$

massively changing the gene expression and the whole balance mechanism.

More comparisons on the difference and usage of metaheuristic methods are available in [30], [31].

B. GENETIC EXPRESSION PROGRAMMING

Genetic expression programming (GEP) [32] is a variation of GP. Given a set of terminals (input variables or constants) and operators (functions), GEP describes programs or formulas as binary trees represented in linear gene expressions. The gene expression is the level-order traversal of a binary tree and is composed of *head* and *tail*. Symbols in the head can represent functions or terminals; the ones in the tail can only be terminals. The length of a gene expression is $h+t$, where h is the length of head given by the users and $t = h \times (n-1)+1$ the length of tail. The parameter n is the number of most arguments of a function, also given by the users. Referring to Fig. 3(a), symbols x, y and integers are terminals and $+, -, \times, \div, \sqrt{}$ are operators. A GEP tree with $h = 7$ and $n = 2$ is presented in Fig. 4, wherein the gene expression of length 15 ($t = 8$) expresses $(3x+8)\times(\sqrt{y}/2)$. Notice that the GEP expression is determined by level-order traversal of its binary tree. Some terminals in the tail part may be useless. They are used as operands only if their parent nodes are operators. The genetic operators in GEP are reproduction, mutation, transposition, and recombination, applying on the gene expression. Reproduction and mutation are the same as those in GA. Transposition is to prune a gene segment and insert it to a position randomly selected. Recombination is similar to crossover in GA by breaking and recombining two gene expressions. The use of expression trees brings efficiency to GEP because genetic operations can be more easily applied in a simple linear structure. The cost for search specific gene elements in a genetic operation is reduced from $O(n)$ to $O(1)$. GEP has superior performance in dealing with complex problems than conventional regression methods; for example, modeling sensor characteristics [33], diagnosis and prediction of lung cancer [34], fault diagnosis of power transformers [35], and intrusion detection of power grid [36]. Refer to [32] for more details about GEP.

IV. THE PROPOSED METHODS

A. SYMBOLIC LOAD MODELS BY GEP

Symbolic regression is to find a description model from the input-output tuples of a problem that can rationally predict the correspondence of a new input. By Eq. (1), suppose the load of a VM v is determined by k parameters of resource settings, $u_1, \ldots, u_k$. Let $U = \{\langle u_{1i}, \ldots, u_{ki}, y_i\rangle\}$ and y_i be the load of v, measured with the settings of $u_1, \ldots, u_k$ at the ith instance. Suppose $\tilde{f}(u_1, \ldots, u_k)$ is a symbolic regression model describing the relation of y and $u_1, \ldots, u_k$ obtained from U. If the difference, usually measured by MSE (mean-square error), between $\tilde{f}(u_{1i}, \ldots, u_{ki})$ and y_i is acceptable, the model can be used for describing and predicting the load of VMs.

$$MSE(\tilde{f}, U) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{f}(u_{1i}, \ldots, u_{ki}))^2, \tag{8}$$

where n is the size of U. When using GEP, terminals are numeric constants and resource parameters, $u_1, \ldots, u_k$, and the operators are mathematic calculations, as listed in Table 1. For obtaining a fast VM migration decision, the operators performing complicated calculations (and consequently require longer computational times) are assigned lower evolutionary priority.

B. VM ASSIGNMENT BY GA

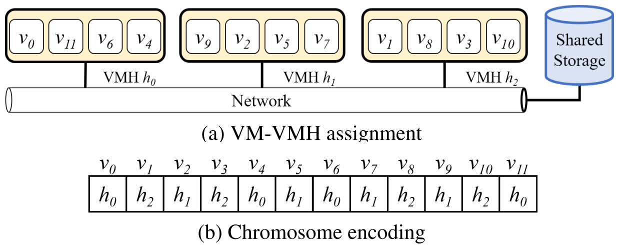
Since there may be many VMs and VMHs working in the cloud, deciding an optimal VM-VMH assignment is combinatorially explosive and time-consuming. Below we show how GA is applied for fast and reliable VM-VMH assignments.

1) CHROMOSOME ENCODING

Suppose that there are q VMs, $v_0, \ldots, v_{q-1}$, served by p VMHs, $h_0, \ldots, h_{p-1}$, in a VMH server farm. A chromosome consisting of q genes is defined as follows.

$$[\varsigma_0, \varsigma_1, \ldots, \varsigma_{q-1}], \tag{9}$$

where ς_i, $0 \le i \le q-1$, denotes that the ith VM is served by the VMH h_i, $\varsigma_i \in \{h_0, h_2, \ldots, h_{p-1}\}$. A chromosome



v_0 v_{11} v_6 v_4
v_9 v_2 v_5 v_7
v_1 v_8 v_3 v_{10}
Shared Storage
VMH h_0
VMH h_1
VMH h_2
Network
(a) VM-VMH assignment
v_0 v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11}
h_0 h_2 h_1 h_2 h_0 h_1 h_0 h_1 h_2 h_1 h_2 h_0
(b) Chromosome encoding

FIGURE 5. **Chromosome encoding for a VM-VMH assignment.**

of the form in Eq. (9) represents a VM-VMH assignment. For example, Fig. 5 presents the assignment of 12 VMs to 3 VMHs.

2) LOAD ESTIMATIONS

For the load balancing of VMHs by VM migration, a good configuration should correct the currently imbalanced situation. It will not result in a new imbalance after the migration of VMs. Suppose the chromosome in Eq. (9) is considered. Let ξ_0 be the VM-VMH assignment before migration and ξ a new assignment generated by GA. Let $\xi(v_i) = h_i$ denote the VMH assigned to v_i in ξ and $\xi^{-1}(h_i) = \{v_j|\xi(v_j) = h_i\}$ the set of VMs assigned to h_i in ξ. By Eq. (2), the load of h_i is estimated as

$$\tilde{L}_H(h_i) = L_0(h_i) + \sum_{j=1}^{*} \tilde{L}_V(v_j), \tag{10}$$

where $v_j \in \xi^{-1}(h_i)$ and $\tilde{L}_V(v_j) = \tilde{f}(u_1, \ldots, u_k)$ is the load predicted by the regression model presented in Eq. (8).

3) MIGRATION COST

The cost of migration is also a critical issue that should be considered. Migration is costly, i.e., the time/resources consumption for retaining and moving VM data and the profit loss due to service suspension. Therefore, for a new VM-VMH assignment, the fewer moving VMs, the better. Let θ be the cost of moving a VM, the migration cost $\alpha(\xi)$ of a chromosome ξ is to examine $\xi(v_i)$, $0 \le i \le q-1$, against ξ_0, i.e.,

$$\alpha(\xi) = 1 - \frac{1}{q}\sum_{i=0}^{q-1} \gamma(v_i), \tag{11}$$

$$\gamma(v_i) = \begin{cases} 0, & \xi(v_i) = \xi_0(v_i), \\ \theta, & \xi(v_i) \neq \xi_0(v_i), \end{cases} \tag{12}$$

where $0 \le \theta \le 1$ indicates the impact of migration of a VM and is defined by the system administrator. In a configuration with 0 migrating VMs, the migration cost is 0; otherwise, the migration cost is accumulated as the number of migrating VMs increases. Migration of VMs is costly, even with shared storage (live-migration). However, migration of VMs is a need for the administration of VMH servers from energy-saving or SLA aspects [37]–[39]. A higher θ setting reduces the occurrence of migration and eases the impact or inconvenience resulted from migration. Setting a reasonable θ can refer to the aspects stated in [24], [40].

4) FITNESS FUNCTION

The fitness of ξ is defined by considering the migration cost $\alpha(\xi)$ and the balance factor $\beta(\xi)$. The balance factor $\beta(\xi)$ considers the load difference of two VMHs and computes the balance ratio against the maximum workload predicted.

$$\beta(\xi) = \ln\left(\frac{\frac{1}{2}p(p-1)\max_{\forall j}(\{\tilde{L}_H(h_j)\})}{\sum_{\forall a \neq b} |\tilde{L}_H(h_a) - \tilde{L}_H(h_b)|}\right). \tag{13}$$

The overall fitness of ξ is defined as follows,

$$\mathit{fitness}(\xi) = \begin{cases} \alpha(\xi)\cdot\beta(\xi), & L_*(h_j) \le \tilde{L}_H(h_j) \le L^*(h_j), \\ & \text{for all } j, \\ 0, & \text{otherwise} \end{cases} \tag{14}$$

Because the load of VMHs is expected to be "balanced" after migration, the fitness of a configuration is bad (0) if any VMH is predicted to be overloaded ($> L^*$) or underloaded ($< L_*$). Otherwise, the migration cost ($\alpha(\xi)$) and the balancing degree ($\beta(\xi)$) are evaluated.

C. LOAD-BALANCING PROCEDURE

Based on the proposed genetic-methods, an intelligent load balancing mechanism is implemented with two main procedures. Suppose there are q VMs running on p VMHs, the parameters associated with the generation of a GEP-based load model include: historical VM load data or the log records periodically collected by VMHs: $\mathbf{D}$, a linear GEP chromosome ζ_0 converted from the current load model, a set of operators: $\mathbf{O}$, population size: k, best GEP trees to retain: t, MSE threshold: ϵ by Eq. (8), and head ($\mathbf{h}$) and the maximum number n of operators in a GEP tree. The pseudocode of the GEP-based load model generation is presented in **Algorithm 1**. The load-balance procedure is invoked periodically with the following parameters. The current VM-VMH assignment $\xi_0 = [\varsigma_0, \varsigma_1, \ldots, \varsigma_{q-1}]$ in the form of Eq. (9), Load-Balance Status (LB) by Eq. (13), Population size: K, Best chromosomes to retain: T, Best fitness threshold: $B = M \times \mathit{fitness}(\xi_0)$. The pseudocode of our proposed GA-based load balance mechanism is presented in **Algorithm 2**.

Notice that the current VM load model ζ_0 and the current VM-VMH configuration ξ_0 are also include the initial population of `GEP_Load_Model_Generator` and `GA_Balance_Procedure`, respectively. Such settings make sure that solutions better than the current one can be derived. Initially, the training data for the VM load model can be manufactured, as shown in Section V-B. They are collected from VMH's log and used to update the VM load model, as shown in `GEP_Load_Model_Generator`. Therefore, as the system performs, the VM load model evolves reasonably to the real situation. `GA_Balance_Procedure` performs load balance when an imbalance is detected. It searches by GA a new VM-VMH assignment better than the current

Algorithm 1 The GEP-Based Load Model Generation

```
1:  function GEP_Load_Model_Generator
2:      Compute tail size $t = h \times (n-1) + 1$;
3:      Evaluate MSE $\epsilon_0$ of $\zeta_0$ against $\mathbf{D}$;
4:      while ($\epsilon_0 > \epsilon$) do
5:          Randomize $k$ GEP chromosomes $\zeta_i$, $1 \le i \le k$, $\zeta_i$ is of length $h + t$ (as in Fig. 4(a));
6:          Set the initial population as $R = \cup_{i=0}^{k}\{\zeta_i\}$;
7:          Evaluate MSE $\epsilon(\zeta_i)$ of $\zeta_i$ in $R$ by Eq.(8);
8:          repeat
9:              Retain top-$T$ elements of $R$ with lower MSE;
10:             Perform GEP operators on $\zeta$'s;
11:             Evaluate MSE $\epsilon(\zeta_i)$ of $\zeta_i$ by Eq.(8);
12:             Select the best element $\zeta'$ as the best VM load model and $\epsilon_0 = \epsilon(\zeta')$;
13:         until (termination conditions: max. iteration or low-enough MSE)
14:     end while
15: return GEP expression of $\zeta'$ as the VM load model;
16: end function
```

Algorithm 2 The GA-Based Load Balance Procedure

```
1:  procedure GA_Balance_Procedure
2:      Evaluate load of each VMH, $L_H(h_i)$, $0 \le i \le p-1$;
3:      By LB, evaluate load balance status al all VMHs;
4:      while (LB $\le 0$ lasts for 30 sec.) do
5:          Randomize $K$ chromosomes, $\xi_i$, $1 \le i \le K$, all in the form of Eq. (9);
6:          Set the initial population as $S = \bigcup_{i=0}^{K}\{\xi_i\}$;
7:          Evaluate the fitness of all $\xi_i$ in $S$ by Eq.(14);
8:          repeat (on $S$)
9:              Retain top-$T$ elements with best fitness;
10:             Perform crossover/mutation on $\xi$'s;
11:             Evaluate the fitness of each $\xi_i$ by Eq.(14);
12:         until (termination conditions: max. iterations, time limitation, high fitness)
13:         Select the best chromosome $\xi'$ as the new VM-VMH assignment if $fitness(\xi') \ge |B|$;
14:         Perform migration by $\xi'$;
15:     end while
16: end procedure
```

configuration and suitably re-balance the system. The two procedures work individually and periodically, as depicted in Fig. 6.

V. EXPERIMENTS

A. ENVIRONMENT AND PARAMETER SETTINGS

The proposed methods were implemented and verified in a small-scale but real cloud environment, Jnet. Jnet is a private intranet, consisting of more than 20 various types of servers accessed by about 200 users. The VMHs and the load balance monitor were attached to Jnet. The experimental environment consisted of four VMH servers, including three HP ProLiant BL460c blades and one HP ProLiant ML350, all running in CentOS. The shared storage server was an HP ProLiant ML110 running FreeNAS. All machines were connected by 1GB ethernet. KVM is the platform for VMHs. The centralized load balancing mechanism was executed and controlled by the VMH0 node. The architecture and specifications are shown in Fig. 7. Because the resource capacity of each VMH is different, the number of VMs that a VMH can support is also different. In the following experiments, the VM capacity of VMH0, VMH1, and VMH2 is 48, and the capacity of VMH3 is 12. There are in total 48 VMs running in each experiment.

A VMH manages the resource consumption of VMs by setting capacity parameters that restrict the number of resources used by VMs. Such parameters are contract-protected and adjustable. In this study, two parameters of CPU utilization are discussed.

- CPU utilization: This parameter explicitly limits the CPU consumption of a VM and restricts the VM's performance/load. There are various implementations of such parameters on different virtualization platforms. The one used in this study is `cpu.cfs_quota_us` defined by the Linux Cgroups [41], which ranges from 1000 to 100000, representing the CPU utilization from 1% to 100%.
- Usage priority: This parameter defines the priorities of VMs for using a CPU core. Also, the parameters `cpu.shares` defined be the Linux Cgroups is used in this study, which ranges from 1 to 65535.

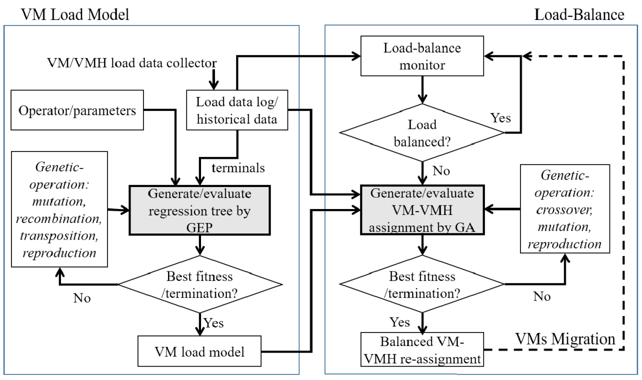

VM Load Model
Load-Balance
VM/VMH load data collector
Operator/parameters
Load data log/ historical data
Genetic-operation: mutation, recombination, transposition, reproduction
terminals
Generate/evaluate regression tree by GEP
Best fitness /termination?
No
Yes
VM load model
Load-balance monitor
Load balanced?
Yes
No
Generate/evaluate VM-VMH assignment by GA
Genetic-operation: crossover, mutation, reproduction
Best fitness /termination?
No
Yes
Balanced VM-VMH re-assignment
VMs Migration

FIGURE 6. Load monitoring and balancing process.

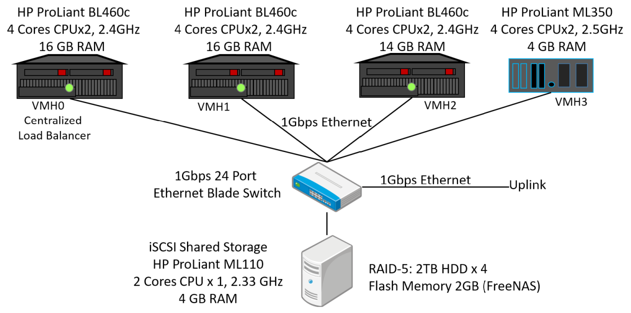


FIGURE 7. Experimental environment settings.

This study defines two parameters for evaluating the performance of the load balancing methods.

- Load-Balance status (LB). LB is the value of Eq. (13) applied to the current VM-VMH configuration, the more balanced load of VMHs, the higher LB.
- Balancing Efficiency (BE). BE is the number of load balancing iterations performed by the monitor to make all VMHs balanced (LB>0) after migration of VMs; the fewer rounds, the faster the load balancing efficiency.

The balancing and monitoring mechanisms were implemented in C++ with libvirt [42] as the software interface. Each VMH scans and logs its CPU load by Eq.(2) every 10 seconds. A load imbalance situation is detected if LB<0 lasts for the past 30 seconds. Load limitations of a VM are $L^* = 25\%$ and $L_* = 0\%$. The length of a chromosome is the number of VMs currently served in the data center. In the following experiments, six load balancing heuristics are compared with our proposed one.

1) LtoS: move the top-loaded VM from over-loaded VMHs to the VMH with the lowest load.
2) LtoR: move the top-loaded VM from over-loaded VMHs to a random VMH.
3) StoS: move the lowest loaded VM from over-loaded VMHs to the VMH with the lowest load.
4) StoR: move the lowest loaded VM from over-loaded VMHs to a random VMH.
5) RtoS: move a random VM from over-loaded VMHs to the VMH with the lowest load.
6) RtoR: move a random VM from over-loaded VMHs to a random VMH.

B. MODELING VM LOADS

The first experiment is to derive load models of VMs. For training GEP, a set of data in the form of $\langle$`cpu.cfs_quota_us`, `cpu.shares`, $L_V(\cdot)\rangle$ was collected. Because there is no historical VM load data, we have to generate load data similar to real VMs. A VM v^* was designed for collecting load data with a load generator. The generator performed several computation-intensive and I/O-intensive instructions. The computation-intensive part is to calculate `sin((double)rand()/RAND_MAX)` and the I/O-intensive part is to read data from `/dev/zero` and write to `/dev/null`. The program was randomly executed and lasted for 10 seconds, the computation or I/O instructions on v^*. The parameters, `cpu.cfs_quota_us` and `cpu.shares` were randomly set for v^*; `cpu.cfs_quota_us` was set 10000–100000 and `cpu.shares` was partitioned into 1024 levels. A total of 1024 combinations of parameters were generated, each used to initialize v^*. The VM v^* executed on a VMH for seven times, and the load $L_V(v^*)$ was measured. A total of 7168 load data were collected. Then, the operators listed in Table 1 and the hyper-parameters listed in Table 2 were used with terminals `cpu.cfs_quota_us` and `cpu.shares` for symbolic regression of $L_V(v^*)$.

TABLE 2. GEP hyper-parameters.

Parameters	Value	Parameters	Value
Max Generations	6000	Population size	30
Best fitness	0.999	Prob. Transposition	2.7%
Gene head length	7	Prob. Recombination	2.7%
Gene tail length	8	Prob. Mutation	2.0%

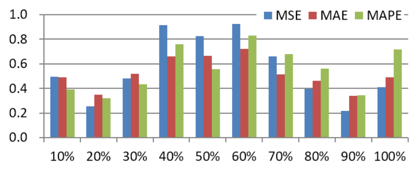


FIGURE 8. GEP training results with 10%–90% training data.

TABLE 3. Comparison with other regression methods.

	GEP	POL	LIN	EXP	POW
OPs	5	11	5	6	**4**
MSE	**0.02408**	0.02470	0.02468	4.85929	0.12902
MAE	**0.11926**	0.12138	0.12250	1.53926	0.26876
MAPE	10.75789	11.14869	21.80033	78.37022	**7.39531**

Boldface: the best case in the row

To verify the effectiveness of the GEP models, some regression methods [43], polynomial(POL), linear(LIN), exponent(EXP), and power(POW) regression, were used to analyze the same VM load data and compared it with GEP. Variable portions of the training data were randomly selected for training, and the rest portions were used for validation. The same training process was repeated ten times. The training results are presented in Fig. 8. Because the best model was with 90% of training data, the same data set was also applied to other regression methods. The load models generated by various regression methods are as follows:

- POL: $-0.166 + 0.0939\, u_1 + 0.0001\, u_1^2 - 0.00004\, u_2 + 0.00000001\, u_2^2$
- LIN: $-0.3235 + 0.103\, u_1 - 0.00002\, u_2$
- EXP: $\exp(-0.4866 + 0.0337\, u_1 - 0.00004\, u_2)$
- POW: $0.0394\, u_1^{1.2216} u_2^{-0.0004}$
- GEP: $\ln((1/u_2) - 0.06\, u_1^2)$

For simplicity, in the above models, `cpu.cfs_quota_us` $= u_1$ and `cpu.shares` $= u_2$. Fig. 9 and Table 3 present the testing results of all regression models. In terms of MSE and MAE, GEP, POL, and LIN perform competitively, and EXP is the worst. MAPE indicates that GEP, POL, and POW are more stable than others. POL and GEP produce similar error rates; however, POL uses more and complicated operators than GEP does. LIN uses the fewest operators but is more unreliable in terms of MSE/MAE and unstable in MAPE than GEP. GEP produces a much compact formula than others and can be considered competitive.

C. LOAD BALANCE SCENARIOS

In the following experiments, the VM load model generated by GEP was used for load prediction. The load balancing

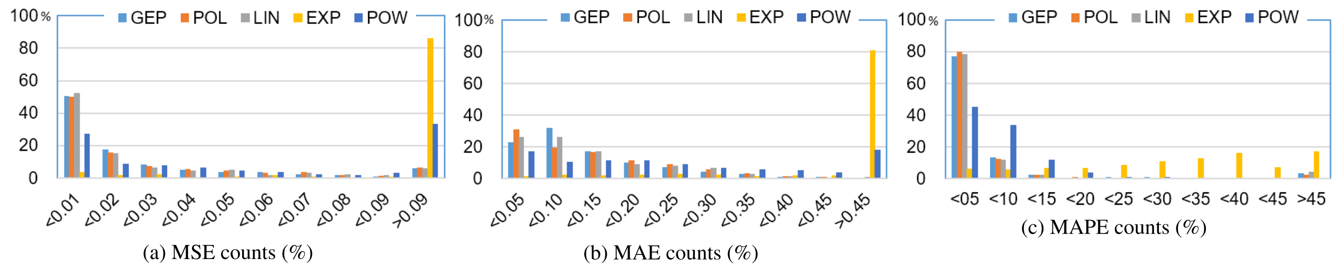


FIGURE 9. Statistics on errors of all models.

TABLE 4. Comparisons on average load (%) in Scenario-A.

	Ours	LtoS	LtoR	StoS	StoR	RtoS	RtoR
VMH0	54.9	43.7	**17.9**	19.2	96.4	86.2	96.1
VMH1	50.0	43.8	95.3	92.4	**3.9**	30.3	11.2
VMH2	53.8	71.9	17.9	19.8	**5.3**	31.2	8.0
VMH3	57.0	37.4	13.8	18.2	**3.2**	24.7	6.6
Average	53.9	49.2	36.2	37.4	**27.2**	43.1	30.5
StdDev	**5.1**	26.8	68.3	63.6	80.0	50.0	75.8

monitor of Fig. 6 was applied to four scenarios. A total of 48 VMs were deployed, each of which performed similarly as v^*. In the following tests, 40 runs of load balancing were observed. VM migration was performed when the load was imbalanced, or the GA produced a VM-VMH assignment with LB 1.25 times better than the current configuration. The hyper-parameters associated with GA include: max. generation: 1000, population size: 30, crossover_rate: 0.7, mutation_rate: 0.4, time limitation: 60 sec.

1) SCENARIO A

In this scenario, an extreme imbalance is simulated. In the beginning, all VMs were gathered together on a randomly selected VMH, making the VMH heavily overloaded. Fig. 10 plots the change of load of all VMHs in 40 rounds of load-balancing. Table 4 lists the loads of all VMHs, where R2R is the average of RtoR1 and RtoR2 (Fig. 10(g)(h)). From Fig. 10, it is observed that in the first round, all VMs are concentrated to a VMH, the VMH loads are extremely imbalanced, and thus, the LB value is 0. However, our proposed method takes only one round to re-balance the loads for all VMHs. The workloads of all VMHs approximate evenly after load-balancing, which can be observed in Table 4. Other strategies take more iterations for re-balancing the loads. In the cases using LtoS, StoS, and RtoS, LtoS migrates the most heavily loaded VMs to the lowest loaded VMH and reaches a balanced state in 20 rounds, causing load vibration before back to balanced. Though RtoS takes 35 rounds for re-balancing and StoS even not back to balanced, the load of VMHs trend to balance. Other strategies even not back to the balanced state.

2) SCENARIO B

Another extremely imbalance is studied in this experiment, wherein a temporary service suspension of a VMH happens

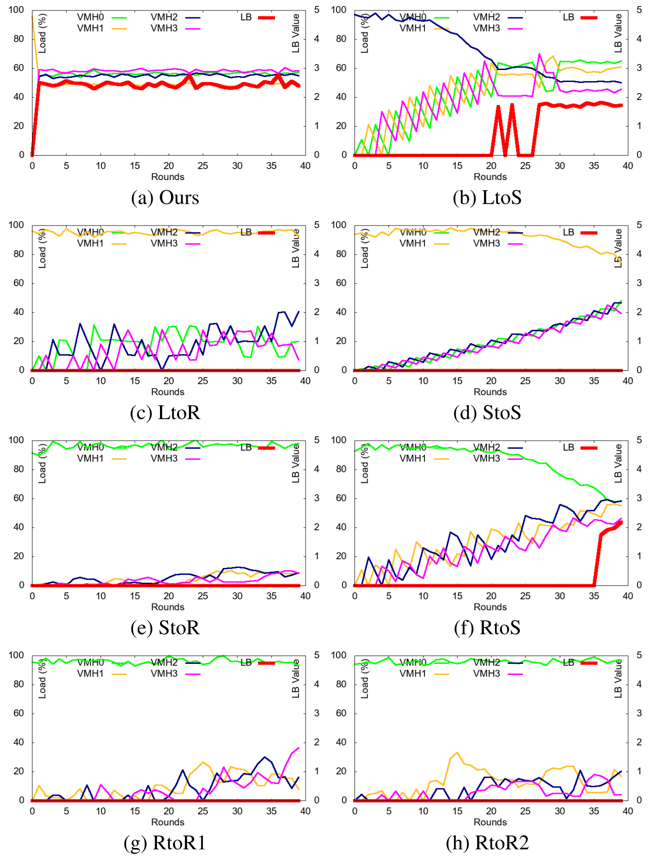


FIGURE 10. Loads and LB trends in Scenario-A.

so that all its VMs have to be migrated to other VMHs for non-stop services. In this scenario, all VMs on VMH1 are moved to VMH0 at round 0, and VMH1 backs to service at round 1. The balancing status is observed in Fig. 11. It is found that VMH0 is overloaded, and VMH1 is idle in the beginning. Again, our proposed method takes only one round to re-balance loads of all VMHs and keep them almost balanced in the consequence rounds. Some strategies can also re-balance the loads, such as LtoS, LtoR, StoS, and even RtoR; however, they take more rounds to re-balance and only manage a low-level balanced status. This phenomenon can be observed in Table 5.

3) SCENARIO C

This experiment simulates the daily use of VMs–the turning on/off of VMs at peak/non-peak hours, wherein more VMs

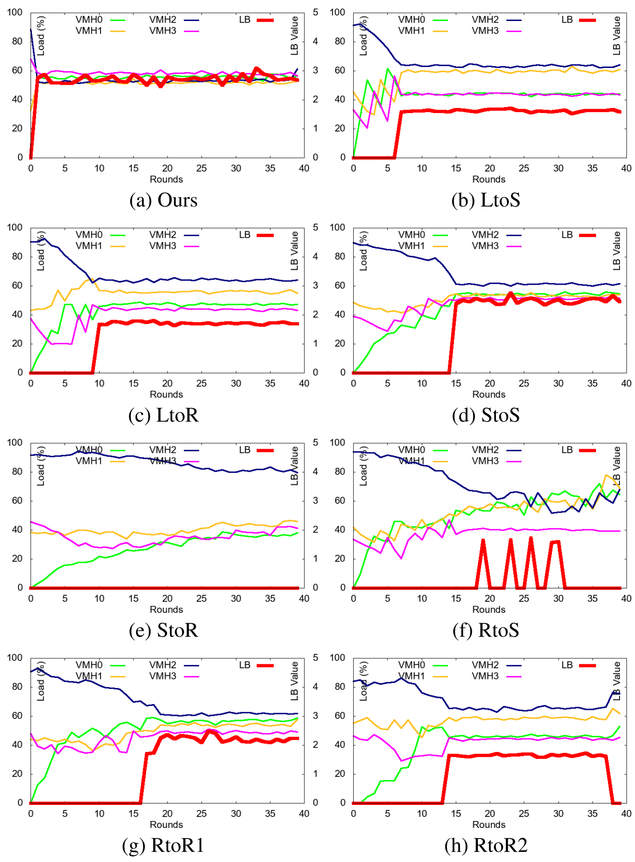


FIGURE 11. **Loads and LB trends in Scenario-B.**

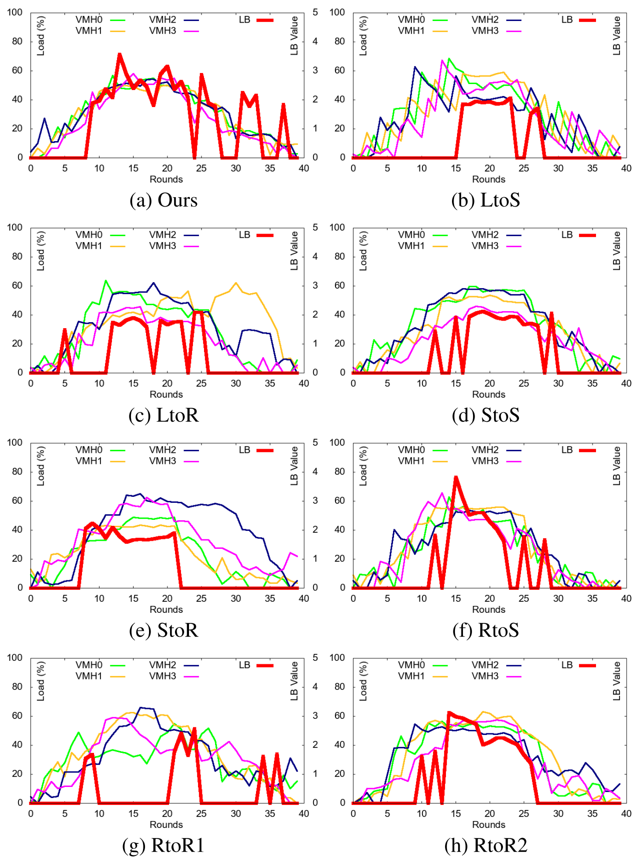


FIGURE 12. **Loads and LB trends in Scenario-C.**

TABLE 5. **Comparisons on average load (%) in Scenario-B.**

	Ours	LtoS	LtoR	StoS	StoR	RtoS	RtoR
VMH0	54.4	43.3	43.1	44.6	**26.9**	50.6	44.5
VMH1	51.2	56.4	55.2	50.6	**40.2**	52.2	53.2
VMH2	**54.0**	66.9	68.3	69.1	86.8	71.6	71.1
VMH3	58.6	42.3	40.2	47.3	**35.2**	37.4	43.6
Average	54.5	52.2	51.7	52.9	**47.3**	53.0	53.1
StdDev	**5.3**	20.3	22.3	19.2	46.6	24.4	22.9

are online/offline in the peak/non-peak hours. Thus, VMHs are less-loaded at non-peak hours and vice versa. The timing of turning on or off a VM is decided by a Gaussian distribution model, with $\mu = 10$ and $\sigma = 5$ for turning on VMs and $\mu = 20$ and $\sigma = 5$ for being online before turning off. That is, a VM starts at around the 10th round and stays online for about 20 rounds, simulating the 20th round as the peak hour and the 1st and 40th rounds the off-peak hours. The results are presented in Fig. 12. Notice that, due to no fair comparison on two VMH load that is stochastic in this scenario, average load of VMHs is not presented. Obviously, the VMH load in this scenario is highly dynamic, which challenges the load-balancers. In the beginning, all methods do not balance VMH load effectively due to the unpredictable behaviors of VMs being online, and all LBs are 0. It is found that all methods can balance loads to some extend in peak hours. This may be because the load behaviors become stable in the peak hours, with more VMs being online so that the load status is more easily to remain after VM migration. The LtoR strategy is the first to re-balance the load; however, it only works in the non-peak hours (the 5th round) when there are only a few VMs. Some strategies, including LtoR, LtoS, StoR, do not balance loads effectively in the peak hours and even failed after the peak hours. Our proposed method demonstrates a better performance on load balancing than others.

4) SCENARIO D

Scenario D is an extended version of Scenario C, wherein the dynamics of VMHs is considered. In this experiment, the system load of a VMH, i.e., $L_0(\cdot)$, bursts intentionally, and the VMH is undoubtedly overloaded. Thus, the VMs served by this VMH have to migrate for load-balancing. Additionally, VMs behave like those in Scenario C; they turn on and off in peak/non-peak times. Two VMHs were randomly selected as the load-burst hosts at the 7th and 27th rounds, respectively. Their load bursts continue for seven rounds and back to service at the 14th and 34th rounds, respectively. The balancing results are presented in Fig. 13. Our proposed method is sensitive to the imbalance in this highly dynamic environment and can re-balance the VMH loads efficiently. Static heuristic rules, such as LtoS and StoS, are also sensitive and able to

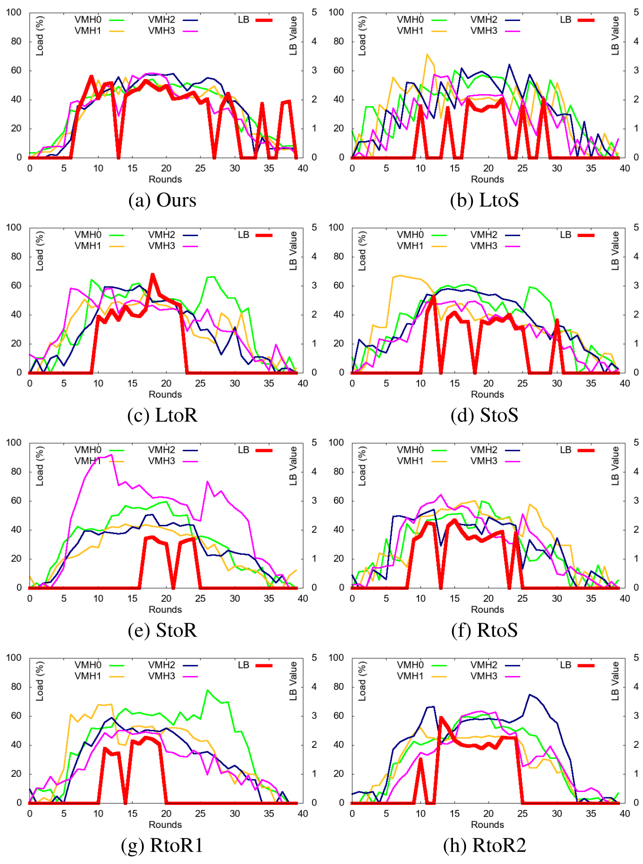


FIGURE 13. Loads and LB trends in Scenario-D.

TABLE 6. Comparisons on LB and BE of all scenarios.

(a) LB

Scenario	Ours	LtoS	LtoR	StoS	StoR	RtoS	RtoR
A	**2.4**	0.7	0.0	0.0	0.0	0.2	0.0
B	**2.7**	1.3	1.3	1.6	0.0	0.2	1.2
C	**1.3**	0.5	0.6	0.7	0.6	0.7	0.6
D	**1.4**	0.5	0.7	0.7	0.3	0.7	0.6
Average	**1.95**	0.75	0.65	0.75	0.23	0.45	0.60
StdDev	0.70	0.38	0.53	0.66	**0.29**	0.29	0.49

(b) BE

Scenario	Ours	LtoS	LtoR	StoS	StoR	RtoS	RtoR
A	**1.0**	12.5	40.0	40.0	40.0	36.0	40.0
B	**1.0**	7.0	10.0	15.0	40.0	4.4	12.5
C	**1.4**	15.0	6.8	5.2	26.0	4.0	6.4
D	**1.7**	4.3	27.0	8.7	16.5	13.0	9.9
Average	**1.28**	9.70	20.95	17.23	30.63	14.35	17.20
StdDev	**0.34**	4.91	15.49	15.72	11.50	15.02	15.40

re-balance; they do not work well in all cases. Random heuristic rules may be workable; they are not stable and have poor performance.

D. SUMMARY

Table 6 lists the averaged LB and BE values of all methods in the four scenarios. A higher LB value means a better-balanced performance. A lower BE value presents better balancing efficiency. Our proposed method demonstrates effectiveness and efficiency in either stable (e.g., scenarios A and B) or dynamic (e.g., C and D) environments in terms of LB and BE. With VM load models obtained from GEP, our method predicts the VMH loads after migration and computes by GA the most effective VM-VMH assignment for migration. Other strategies perform heuristic VM migration and take longer rounds for re-balancing, or even not back to balance. Interestingly, according to LB and BE in Table 6, LtoS is the second-best strategy. LtoS is intuitive and straightforward to implement, i.e., to move the worst VM to a VMH with the lowest load; however, it takes a longer time to re-balance VMH loads. Random methods, such as LtoR, StoR, RtoS, and RtoR, are effective sometimes but are not recommended due to their stochastic characteristics and unstable performance. Our proposed method performs effective and efficient load balancing in all tests. It is competitive and more promising than others.

`GEP_Load_Model_Generator` updates the VM load model every 120 seconds periodically. The monitor `GA_Balance_Procedure` is activated every 250 seconds and calculates the LB value of all VMHs. Notice that the frequency of executing the two procedures should not be too high. Higher updating and balancing frequency may keep the VM load model and the system as real and balance as possible. However, the operations of performance measurement and migration are costly: the steal of CPU cycles degrades the overall performance and QoS, whereas frequent migration may cause unacceptable service suspension. Hence, the period of load monitoring is 250 seconds and the period of updating the VM load model is 120 seconds.

Moreover, as in other machine learning algorithms, setting hyper-parameters associated with GA and GEP is subtle and complex. The size of population, probabilities of genetic operators, and the termination condition are all problem-dependent. In general, higher probabilities of invoking genetic operators usually drive GA or GEP early mature but easily trapped in local optimal; lower probabilities take much time to converge. A large population consumes considerable memory space and computing time but may earn a higher chance of evolving a better solution. Due to the costs of performance probing and migration, deriving a solution (VM load model or VM-VHM assignment) should be timely. Such a solution may not be the best one but has to be acceptable and better than the current one. For example, a new VM-VMH assignment has to be 1.25 times better than the current one to be acceptable. Other hyper-parameters are set by observing the behaviors of VMs and VMHs in JNet. Several combinations of parameters are attempted. For the length of this paper, some meaningful parameter sets are presented in the experiments. For issues of setting hyper-parameters associated with genetic methods, the readers can refer to [44], [45].

Regarding the efficiency of the balance mechanism, the main concern is how fast an effective VM-VMH assignment is suggested. The complexity of each strategy depends on how it is implemented. In this study, all strategies are implemented with standard C/C++ libraries. Searching for a min/max item can be done in $\mathbf{O}(n)$ and generating a random number can be done in $\mathbf{O}(1)$, where n is the number of VMs. However, running a GA-based method is stochastic. Its complexity depends on the representation of chromosomes, the implementation of the genetic operators, and the fitness function. Some studies presented theoretic and experimental analysis on the complexity of GA [46], [47]. The complexity of our method is more likely $\mathbf{O}(gpn)$ for the stochastic process and $\mathbf{O}(n^2)$ for fitness evaluation, where g is the number of generations and p is the population size. In our experiment, g is 1000 and p is 30. The GA iteration is usually terminated by a low LB value and is less than g. When implementing our method in a large-scale cloud environment, the fitness evaluation could be the computational bottleneck, which can be implemented with a sophisticated data structure to reduce the computation cost.

VI. CONCLUSION

This paper presents a load balancing mechanism based on evolutionary computing. Loads of VMs and the associated resource parameters are measured and used for constructing symbolic regression models of VM using GEP. An optimal combination of VM-VMH assignment is decided by GA, which predicts VMH loads by GEP models and suggests the VMs be migrated for load-balancing. Experiments are conducted in a small-scale but real cloud environment. The proposed method demonstrates its effectiveness and efficiency on load balancing and is competitive and promising. Although the proposed method performs well for load-balance, it can be improved in some aspects and discussed.

Currently, constant lengths of the head and tail of a GEP chromosome may limit the searchability of GEP in the solution space. The variable size of the head that is determined by the training data may produce better regression models. The initial VM load model is built from simulated data. However, according to the architecture depicted in Fig. 6, VM load data are collected continuously; the VM load model can also be updated accordingly. GEP and GA are the genetic-based methods employed in this study. It is possible to use other methods to generate regression models and decide the optimal combination of VMs for migration, provided that white-box regression models and fast decision-making can be derived.

Due to the hardware resource limitations, the test environment only consists of four VMHs with a centralized load-balancer installed on one VMH. However, the usability of our proposed method is verified. The proposed method can be applied to a distributed, large-scale management architecture easily. At present, the CPU load of VMs and VMHs is considered. Other types of load, like networking bandwidth, memory data transfer rate, or GPU loads, may also be worth studying. However, these types of loads are stochastic, and their measurements take considerable overhead. They are not included in this paper but may be processed similarly by our proposed method.

This study focuses on the load balancing of VMHs that demand VHSs to work evenly. There may be other VMH management strategies, e.g., power saving or utilization maximization, that can be done by load consolidation, i.e., to aggregate VMs to some high-performance or energy-efficient VMHs so that some VMHs can be turned off [48]–[50]. These can be easily done using our proposed methods by re-defining the fitness function of GA. The current fitness function and balance factor are based on the ratio of the maximum load to the load differences among VMHs, which is simple but fast. Comprehensive statistics on the VMH loads may produce a sophisticated fitness function and LB formula [51]. Considering all these factors for load balancing is a multi-objective optimization problem, which requires sophisticated methods for maintaining computation efficiency. However, measurements on the detailed resource consumption of VMs may take much more computational resources and thus degrade the performance of VMHs and the load-balancer. The trade-offs between efficiency and measurement need advanced studies. Instead of GA, other metaheuristic algorithms may also decide VM-VMH assignment efficiently. GA is competitive than others in the practical administration of cloud computing servers because it can easily and flexibly adjust the optimization objectives and maintain a clear and intuitive representation for VM-VMH assignment. Studies on the performance and applicability of various metaheuristic algorithms for different balance purposes are worthy. These will be part of our future work

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[2] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, and A. Ghalsasi, "Cloud computing—The business perspective," *Decis. Support Syst.*, vol. 51, no. 1, pp. 176–189, 2011.

[3] P. Mell and T. Grance, "The NIST definition of cloud computing," Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. 800-145, Sep. 2011.

[4] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the Internet impasse through virtualization," *Computer*, vol. 38, no. 4, pp. 34–41, Apr. 2005.

[5] J. E. Smith and R. Nair, "The architecture of virtual machines," *Computer*, vol. 38, no. 5, pp. 32–38, 2005.

[6] E. J. Ghomi, A. M. Rahmani, and N. N. Qader, "Load-balancing algorithms in cloud computing: A survey," *J. Netw. Comput. Appl.*, vol. 88, pp. 50–71, Jun. 2017.

[7] R. Yadav and W. Zhang, "MeReg: Managing energy-SLA tradeoff for green mobile cloud computing," *Wireless Commun. Mobile Comput.*, vol. 2017, pp. 1–11, Jan. 2017.

[8] M. Ala'Anzy and M. Othman, "Load balancing and server consolidation in cloud computing environments: A meta-study," *IEEE Access*, vol. 7, pp. 141868–141887, 2019.

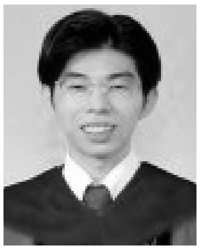
[9] S. Chhabra and A. K. Singh, "Dynamic hierarchical load balancing model for cloud data centre networks," *Electron. Lett.*, vol. 55, no. 2, pp. 94–96, Jan. 2019.

[10] A. Asghar, H. Farooq, and A. Imran, "Concurrent optimization of coverage, capacity, and load balance in HetNets through soft and hard cell association parameters," *IEEE Trans. Veh. Technol.*, vol. 67, no. 9, pp. 8781–8795, Sep. 2018.
[11] H. Yao, X. Yuan, P. Zhang, J. Wang, C. Jiang, and M. Guizani, "Machine learning aided load balance routing scheme considering queue utilization," *IEEE Trans. Veh. Technol.*, vol. 68, no. 8, pp. 7987–7999, Aug. 2019.
[12] J. Li, G. Luo, N. Cheng, Q. Yuan, Z. Wu, S. Gao, and Z. Liu, "An end-to-end load balancer based on deep learning for vehicular network traffic control," *IEEE Internet Things J.*, vol. 6, no. 1, pp. 953–966, Feb. 2019.
[13] R. Alonso-Calvo, J. Crespo, M. García-Remesal, A. Anguita, and V. Maojo, "On distributing load in cloud computing: A real application for very-large image datasets," *Procedia Comput. Sci.*, vol. 1, no. 1, pp. 2669–2677, May 2010.
[14] S. H. Doong, C. C. Lai, S. J. Lee, C. S. Ouyang, and C. H. Wu, "Performance modeling of virtual machines hosted on Xen," in *Proc. Int. Conf. Cloud Comput. Virtualization*, May 2010, pp. 115–122, doi: 10.5176/978-981-08-5837-7_223.
[15] J. Hu, J. Gu, G. Sun, and T. Zhao, "A scheduling strategy on load balancing of virtual machine resources in cloud computing environment," in *Proc. 3rd Int. Symp. Parallel Archit., Algorithms Program.*, Dec. 2010, pp. 89–96.
[16] D. Ardagna, S. Casolari, M. Colajanni, and B. Panicucci, "Dual time-scale distributed capacity allocation and load redirect algorithms for cloud systems," *J. Parallel Distrib. Comput.*, vol. 72, no. 6, pp. 796–808, Jun. 2012.
[17] S. Pang, W. Li, H. He, Z. Shan, and X. Wang, "An EDA-GA hybrid algorithm for multi-objective task scheduling in cloud computing," *IEEE Access*, vol. 7, pp. 146379–146389, 2019.
[18] Y. Dong, G. Xu, Y. Ding, X. Meng, and J. Zhao, "A 'joint-me' task deployment strategy for load balancing in edge computing," *IEEE Access*, vol. 7, pp. 99658–99669, 2019.
[19] M. Gamal, R. Rizk, H. Mahdi, and B. E. Elnaghi, "Osmotic bio-inspired load balancing algorithm in cloud computing," *IEEE Access*, vol. 7, pp. 42735–42744, 2019.
[20] G. J. Popek and R. P. Goldberg, "Formal requirements for virtualizable third generation architectures," *Commun. ACM*, vol. 17, no. 7, pp. 412–421, Jul. 1974.
[21] VMware Inc. *vSphere Hypervisor*. Accessed: Jan. 27, 2021. [Online]. Available: https://www.vmware.com/products/vsphere-hypervisor/
[22] Kernel Based Virtual Machine. *Open Virtualization Alliance*. Accessed: Jan. 27, 2021. [Online]. Available: http://www.linux-kvm.org/page/Main_Page
[23] Hyper-V. *Overview Microsoft*. Accessed: Jan. 27, 2021. [Online]. Available: http://technet.microsoft.com/library/hh831531
[24] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of virtual machine live migration in clouds: A performance evaluation," in *Cloud Computing*, M. G. Jaatun, G. Zhao, and C. Rong, Eds. Berlin, Germany: Springer, 2009, pp. 254–265.
[25] R. Achar, P. S. Thilagam, N. Soans, P. V. Vikyath, S. Rao, and A. M. Vijeth, "Load balancing in cloud based on live migration of virtual machines," in *Proc. Annu. IEEE India Conf. (INDICON)*, Dec. 2013, pp. 1–5.
[26] X. Song, Y. Ma, and D. Teng, "A load balancing scheme using federate migration based on virtual machines for cloud simulations," *Math. Problems Eng.*, vol. 2015, pp. 1–11, Mar. 2015.
[27] C. C. Lai, S. H. Doong, and C. H. Wu, "Machine learning," in *Wiley Encyclopedia of Computer Science and Engineering*. Hoboken, NJ, USA: Wiley, 2008.
[28] A. Fraser, "Simulation of genetic systems by automatic digital computers," *Austral. J. Biol. Sci.*, vol. 10, no. 4, pp. 484–491, 1957.
[29] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.
[30] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Comput. Surv.*, vol. 35, no. 3, pp. 268–308, Sep. 2003.
[31] M. Abdel-Basset, L. Abdel-Fatah, and A. K. Sangaiah, "Metaheuristic algorithms: A comprehensive review," in *Computational Intelligence for Multimedia Big Data on the Cloud With Engineering Applications* (Intelligent Data-Centric Systems), A. K. Sangaiah, M. Sheng, and Z. Zhang, Eds. New York, NY, USA: Academic, 2018, ch. 10, pp. 185–231.
[32] C. Ferreira, "Gene expression programming: A new adaptive algorithm for solving problems," *Complex Syst.*, vol. 13, no. 2, pp. 87–129, 2001. [Online]. Available: http://arxiv.org/abs/cs/0102027
[33] F. M. Janeiro, J. Santos, and P. M. Ramos, "Gene expression programming in sensor characterization: Numerical results and experimental validation," *IEEE Trans. Instrum. Meas.*, vol. 62, no. 5, pp. 1373–1381, May 2013.
[34] H. Azzawi, J. Hou, Y. Xiang, and R. Alanni, "Lung cancer prediction from microarray data by gene expression programming," *IET Syst. Biol.*, vol. 10, no. 5, pp. 168–178, Oct. 2016.
[35] H. Malik and S. Mishra, "Application of gene expression programming (GEP) in power transformers fault diagnosis using DGA," *IEEE Trans. Ind. Appl.*, vol. 52, no. 6, pp. 4556–4565, Nov. 2016.
[36] S. Deng, A. Zhou, D. Yue, B. Hu, and L. Zhu, "Distributed intrusion detection based on hybrid gene expression programming and cloud computing in a cyber physical power system," *IET Control Theory Appl.*, vol. 11, no. 11, pp. 1822–1829, Jul. 2017.
[37] W. Dargie, "Estimation of the cost of VM migration," in *Proc. 23rd Int. Conf. Comput. Commun. Netw. (ICCCN)*, Aug. 2014, pp. 1–8.
[38] A. Choudhary, M. C. Govil, G. Singh, L. K. Awasthi, E. S. Pilli, and D. Kapil, "A critical survey of live virtual machine migration techniques," *J. Cloud Comput.*, vol. 6, no. 1, pp. 1–41, Dec. 2017.
[39] C. Canali, L. Chiaraviglio, R. Lancellotti, and M. Shojafar, "Joint minimization of the energy costs from computing, data transmission, and migrations in cloud data centers," *IEEE Trans. Green Commun. Netw.*, vol. 2, no. 2, pp. 580–595, Jun. 2018.
[40] L. Sliwko and V. Getov, "Transfer cost of virtual machine live migration in cloud systems," Univ. Westminster, London, U.K., Tech. Rep. 1-21, 2017.
[41] cgroups. (2015). *kernel.org*. Accessed: Jan. 27, 2021. [Online]. Available: https://www.kernel.org/doc/html/latest/admin-guide/cgroup-v2.html
[42] libvirt. (2005). *Red Hat*. Accessed: Jan. 27, 2021. [Online]. Available: http://libvirt.org/
[43] N. Draper and H. Smith, *Applied Regression Analysis* (Wiley Series in Probability and Mathematical Statistics). New York, NY, USA: Wiley, 2014.
[44] A. Aleti and I. Moser, "A systematic literature review of adaptive parameter control methods for evolutionary algorithms," *ACM Comput. Surv.*, vol. 49, no. 3, pp. 1–35, Dec. 2016.
[45] N. Dang and C. Doerr, "Hyper-parameter tuning for the $(1 + (\lambda, \lambda))$ GA," in *Proc. Genet. Evol. Comput. Conf.*, Prague, Czech Republic, Jul. 2019, pp. 889–897.
[46] P. S. Oliveto, J. He, and X. Yao, "Time complexity of evolutionary algorithms for combinatorial optimization: A decade of results," *Int. J. Autom. Comput.*, vol. 4, no. 3, pp. 281–293, Jul. 2007.
[47] P. S. Oliveto and C. Witt, "Improved time complexity analysis of the simple genetic algorithm," *Theor. Comput. Sci.*, vol. 605, pp. 21–41, Nov. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0304397515000298
[48] R. Yadav, W. Zhang, K. Li, C. Liu, M. Shafiq, and N. K. Karn, "An adaptive heuristic for managing energy consumption and overloaded hosts in a cloud data center," *Wireless Netw.*, vol. 26, no. 3, pp. 1905–1919, Apr. 2020.
[49] R. Yadav, W. Zhang, O. Kaiwartya, P. R. Singh, I. A. Elgendy, and Y.-C. Tian, "Adaptive energy-aware algorithms for minimizing energy consumption and SLA violation in cloud computing," *IEEE Access*, vol. 6, pp. 55923–55936, 2018.
[50] R. Yadav, W. Zhang, H. Chen, and T. Guo, "MuMs: Energy-aware VM selection scheme for cloud data center," in *Proc. 28th Int. Workshop Database Expert Syst. Appl. (DEXA)*, 2017, pp. 132–136.
[51] N. S. Dey and T. Gunasekhar, "A comprehensive survey of load balancing strategies using Hadoop queue scheduling and virtual machine migration," *IEEE Access*, vol. 7, pp. 92259–92284, 2019.

LUNG-HSUAN HUNG was born in 1985. He received the M.S. degree from the Department of Electrical Engineering, National University of Kaohsiung, Taiwan, in 2014. His research interests include cloud computing and genetic algorithms.

CHIH-HUNG WU (Member, IEEE) was born in 1967. He received the B.S. degree in engineering science from National Cheng-Kung University, Tainan, Taiwan, in 1990, and the M.S. and Ph.D. degrees in electronic engineering from National Sun Yat-sen University, Kaohsiung, Taiwan, in 1992 and 1996, respectively. He is currently a Professor with the Department of Electrical Engineering, National University of Kaohsiung, Kaohsiung. He is also the Director of the Intelligent Computation and Applications Laboratory and the CEO of the Artificial Intelligence Research Center, National University of Kaohsiung. His research interests include artificial intelligence, soft computing, robotics, and cloud computing.

CHIUNG-HUI TSAI was born in 1970. He is currently pursuing the Ph.D. degree with the Department of Electrical Engineering, National University of Kaohsiung, Taiwan. His research interests include machine learning and neural networks.



HSIANG-CHEH HUANG (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electronics engineering from National Chiao Tung University, Taiwan, in 1995, 1997, and 2001, respectively. He is currently a Professor with the Department of Electrical Engineering, National University of Kaohsiung, Taiwan. His current research interests include pattern recognition and image processing.