Hindawi Publishing Corporation
Mobile Information Systems
Volume 2016, Article ID 4859862, 9 pages
http://dx.doi.org/10.1155/2016/4859862

Research Article

Power-Aware Resource Reconfiguration Using Genetic Algorithm in Cloud Computing

Li Deng,[1,2] Yang Li,[1,2] Li Yao,[1,2] Yu Jin,[1,2] and Jinguang Gu[1,2]

[1]*College of Computer Science and Technology, Wuhan University of Science and Technology, Wuhan 430065, China*
[2]*Hubei Province Key Laboratory of Intelligent Information Processing and Real-Time Industrial System, Wuhan, China*

Correspondence should be addressed to Li Deng; dengli@wust.edu.cn

Received 23 September 2016; Accepted 12 December 2016

Academic Editor: Qingchen Zhang

Cloud computing enables scalable computation based on virtualization technology. However, current resource reallocation solution seldom considers the stability of virtual machine (VM) placement pattern. Varied workloads of applications would lead to frequent resource reconfiguration requirements due to repeated appearance of hot nodes. In this paper, several algorithms for VM placement (multiobjective genetic algorithm (MOGA), power-aware multiobjective genetic algorithm (pMOGA), and enhanced power-aware multiobjective genetic algorithm (EpMOGA)) are presented to improve stability of VM placement pattern with less migration overhead. The energy consumption is also considered. A type-matching controller is designed to improve evolution process. Nondominated sorting genetic algorithm II (NSGAII) is used to select new generations during evolution process. Our simulation results demonstrate that these algorithms all provide resource reallocation solutions with long stabilization time of nodes. pMOGA and EpMOGA also better balance the relationship of stabilization and energy efficiency by adding number of active nodes as one of optimal objectives. Type-matching controller makes EpMOGA superior to pMOGA.

1. Introduction

Cloud computing [1] provides a huge resource pool shared by a large number of users. Virtualization technology enables dynamic resource configuration according to real demands of applications [2] and live migration of VMs is an important way to implement resource reallocation in the cloud [3].

Wrasse [4] is designed to handle generalized resource allocation in the cloud. It uses massive parallelism by orchestrating a large number of light-weight GPU threads to explore the search space in parallel. Server consolidation [5–7] has always been studied for green computing. Constraint programming is used to reduce the number of active physical nodes for energy efficiency while the Service Level Agreement (SLA) is guaranteed. Efficient VM migration and placement are also helpful for reducing the number of active PMs. Furthermore, economic efficiency of cloud computing has been studied by many researchers [8, 9]. Auction approaches are presented to balance the relationship between economic efficiency and computational efficiency.

However, current resource management methods seldom consider stability of VM placement globally to improve resource efficiency [10]. Due to time-varying resource demands of applications, current mapping of VMs to physical nodes may be not suitable for future workloads. New hot nodes would appear in the near future, which directly results in another resource reallocation. Resource reallocation would subsequently lead to some additional overheads [11], such as migration time, downtime, and service degradation. The stability of a VM placement pattern should be considered during dynamic resource configuration.

Resource allocation problem is a kind of combinatorial problem, known as NP-hard problem [12]. Evolutionary computation algorithm can approximate an optimal solution only taking polynomial time [12]. In this paper, we present several genetic algorithms for resource allocation in cloud computing based on our prior works [10]. According to prediction information of application workloads, these algorithms all provide resource reconfiguration solutions with long stabilization time of nodes. Our contributions are

listed in the following: (1) we design genetic algorithms to better balance the relationship between node stabilization and power efficiency; (2) a type-matching controller is proposed to accelerate evolution process; (3) we implement genetic algorithms and a type-matching controller in Java and compare the performances of these genetic algorithms.

The rest of the paper is organized as follows: Section 2 discusses related work about dynamic resource allocation. In Section 3, we give the description of problem formulation. Objectives and constraints of dynamic resource allocation are formulated. Section 4 introduces the details of several genetic algorithms. Performance evaluation of several algorithms is done in Section 5. Finally, we give our summary and future research directions in Section 6.

2. Related Work

Being completely different from traditional static resource configuration, cloud computing enables dynamic resource allocation based on time-varying workloads of applications. Resource efficiency is thus improved significantly. Many researchers have studied resource reallocation problems.

Dynamic resource allocation usually has the following objectives.

(i) Green Computing. Energy consumption is the most critical problem in cloud computing [13]. It becomes more serious especially in multicore era [14]. Server consolidation [5, 6, 15] is used to decrease the number of active physical nodes. Power efficiency is greatly improved. Constraints programming [5] and genetic algorithm [15] are, respectively, employed to find a solution using the minimum number of active nodes for green computing. An energy-efficient resource allocation framework [7] is proposed to minimize physical node overload occurrences for overcommitted clouds by predicting future resource utilizations of scheduled VMs.

(ii) Resource Fairness. Resource in the cloud is shared among a large number of tenants. Resource fairness among numerous users is then studied [16, 17]. A multiresource allocation mechanism (called `DRFH`) [16] is presented to ensure fair usage of resource among cloud users using heuristics.

(iii) Resource Efficiency. Resource efficiency becomes very important in large-scale datacenters with tens of thousands of servers [18, 19]. Some approaches are designed to improve computing resource utilization, such as memory [20] and I/O [21]. There are some methods presented to improve SLAs of applications [22]. Also, some resource management solutions are proposed for special applications: stream processing [23, 24] and business process [25].

(iv) Economic Efficiency. Resource in the cloud is usually rent in a pay-as-you-go model. Economic efficiency of cloud computing has been studied by many researchers [8, 9]. Trading mechanisms for the demand response are designed to achieve the maximum social welfare with arbitrarily high probability.

In this paper, our work mainly focuses on the stability of VM placement pattern. Because workloads of applications are time-varying especially in mobile cloud computing, the stability becomes more important.

3. Problem Formulation

Due to dynamic workloads, resource demands of applications vary with time. Some nodes have frequent resource contention and become busy when workloads increase. These nodes are called *hot nodes*. Hot nodes should be alleviated by decreasing their workloads to ensure service level objectives (SLAs) of applications.

Live migration of virtual machine is an important method to alleviate hot nodes. It redistributes VMs on a pool of nodes. When remapping VMs to nodes, we should consider future trends of application workloads to avoid "thrashing," much more hot nodes arising in the future. So, stability is an important metric to choose new VM distribution on nodes. The stability of VM distribution mainly depends on the total workloads of each node.

Abbreviations lists the definition of some symbols used in our discussion.

We have the following equations:

$$y_i = \begin{cases} 0, & \text{if } \sum_{j=1}^{\mathcal{N}} x_{ij} = 0, \\ 1, & \text{if } \sum_{j=1}^{\mathcal{N}} x_{ij} \neq 0, \end{cases} \quad i = 1, \ldots, \mathcal{M},$$

$$m_j = \begin{cases} 0, & \text{if } x_{ij} = x'_{i'j} = 1,\ i = i',\ i, i' = 1, \ldots, \mathcal{M}, \\ 1, & \text{if } x_{ij} = x'_{i'j} = 1,\ i \neq i',\ i, i' = 1, \ldots, \mathcal{M}, \end{cases} \quad j = 1, \ldots, \mathcal{N}. \tag{1}$$

Variable x_{ij} denotes node i hosting VM j in old VM placement pattern $\mathcal{D}$, while $x'_{i'j}$ means that VM j resides on node i' in new VM placement pattern $\mathcal{D}'$.

Some definitions are given in Abbreviations.

Definition 1. A *placement pattern* $\mathcal{D}_k$ is the mode in which a group of applications (VMs) are distributed on physical nodes.

Definition 2. The node i is stable if and only if the node has enough resources for applications (VMs) residing on it during a certain period of time, no matter how the workloads of applications vary.

Definition 3. The placement pattern $\mathcal{D}_k$ is stable if and only if each node in the placement pattern is stable during a period of time.

Definition 4. Stabilization time T means the longest period in which a node or a placement pattern stays stable from a certain time. It is a straight-forward metric to measure the stability of a node or a placement pattern. The stabilization

time of a placement pattern depends on that of each node in it, as shown in the following formula:

$$T_{\mathscr{D}_k} = \min\left\{T_{\text{node}_1}, T_{\text{node}_2}, \ldots, T_{\text{node}_{\mathscr{M}}}\right\}. \quad (2)$$

Then, the problem of dynamic resource allocation is formulated as follows: having known dynamic workloads of VMs (including predicted future workloads), given a set of nodes, the objective of dynamic resource allocation is to find a placement solution of VMs on physical nodes with longest stabilization time, minimal number of VM migration, and minimal number of active nodes:

$$\begin{aligned} \text{Objectives:}\quad & \max \quad T_{\mathscr{D}_k}; \\ & \min \quad \sum_{j=1}^{\mathscr{N}} m_j; \\ & \min \quad \sum_{i=1}^{\mathscr{M}} y_i \end{aligned} \quad (3)$$

$$\text{Subject To:}\quad \sum_{i=1}^{\mathscr{M}} x_{ij} = 1, \quad j = 1, \ldots, \mathscr{N} \quad (4)$$

$$\mathscr{C}_i \geq \sum_{j=1}^{\mathscr{N}} x_{ij}\mathscr{C}'_j, \quad i = 1, \ldots, \mathscr{M} \quad (5)$$

$$Mem_i \geq \sum_{j=1}^{\mathscr{N}} x_{ij} Mem'_j, \quad i = 1, \ldots, \mathscr{M} \quad (6)$$

$$x_{ij}, m_j, y_i \in \{0, 1\}, \quad i = 1, \ldots, \mathscr{M},\ j = 1, \ldots, \mathscr{N}. \quad (7)$$

We have three objectives: one is to make the new distribution of VMs with longest stabilization time ($\max T_{\mathscr{D}_k}$); one is to only migrate the minimal number of VMs from current status to new status ($\min \sum_{j=1}^{\mathscr{N}} m_j$); the last one is to use the smallest number of physical nodes. The first objective means that hot nodes would not appear in the new mapping in a short time. The second objective requests that migration overhead of VMs from old status to new status is minimal. The third objective is to make the number of active physical nodes as small as possible for energy efficiency.

In the above formulae, formula (4) indicates that each VM only resides on one physical node. Formula (5) means that the total amount of CPU resource requested by VMs residing on the same node is not larger than the amount of resource supplied by the node. Formula (6) denotes that the total amount of memory requested by VMs is not larger than the amount of memory supplied by the node. Formula (7) explains that x_{ij}, m_j, and y_i are binary variables.

4. Resource Reconfiguration Approach

As dynamic resource allocation problem is a kind of NP-complete problem, it is hard to find the optimal solution in polynomial time. Using evolution theory of biosphere, genetic algorithm can find an approximately optimal solution to resource allocation problem through simulating biologic evolution process.

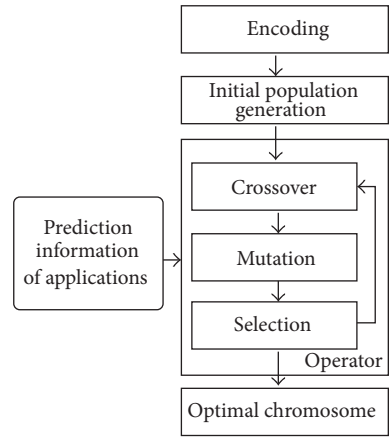
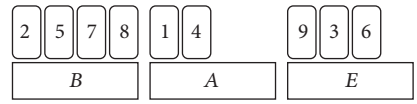


Figure 1: Flow chart of genetic algorithm.



Chromosome: *BEA* (*B*{2, 5, 7, 8}, *E*{1, 4}, *A*{9, 3, 6})

Figure 2: Examples of group encoding scheme.

We propose three algorithms: multiobjective genetic algorithm (MOGA), power-aware multiobjective genetic algorithm (pMOGA), and enhanced power-aware multiobjective genetic algorithm (EpMOGA). MOGA only aims at two objectives: long stability time of VM distribution and minimal number of VM migration. Different from MOGA, pMOGA adds a new objective to be optimized for energy efficiency, shown as formula (3). EpMOGA introduces a type-matching controller based on pMOGA. The type-matching controller is designed to speed up evolution process by matching the type of genes.

4.1. Key Parts of Genetic Algorithm. There are several key parts in genetic algorithm: encoding, initial population generation, main operators (crossover, mutation, and selection), and termination condition, as shown in Figure 1. MOGA, pMOGA, and EpMOGA have the same encoding, the same initial population generation, and the same termination condition.

Encoding. Encoding is to express chromosomes, genes with elements of resource allocation problem. There are three methods to express bin packing problems in genetic algorithm: one gene per object, one gene per bin, and one gene per group (bin and objects in it) [26]. The encoding scheme based on group is employed because it can exactly express the relationship between VMs and physical nodes.

Figure 2 lists examples of encoding scheme using group. In Figure 2, nine VMs are deployed on three nodes. Accordingly, there are three genes in the form of chromosome. Each

gene includes one physical node and several VMs residing on it. A chromosome or an individual signifies a possible solution, a mapping between virtual machines and physical nodes.

Initial Population Generation. A population is a set of chromosomes. Let the population size be *popSIZE*. Genetic algorithm usually starts from an initial population which is often generated randomly. Random generation provides wide search space to find a solution, but it takes much time to get an optimal global solution. First-fit heuristic is used to generate the first population. Note that each individual should meet the constraints discussed in Section 3.

Termination Condition. We set value of the maximum generation (*MAX_GEN*). Iterations would stop when the maximum generation (*MAX_GEN*) is reached.

The difference of the three algorithms mainly lies in operator crossover, mutation, and selection. The difference is discussed below.

4.2. Multiobjective Genetic Algorithm (MOGA). Multiobjective genetic algorithm only has two objectives: long stability time of VM placement and small number of VM migration.

Three main operators (crossover, mutation, and selection) in genetic algorithm are discussed in the following.

Crossover. Crossover is for two parents to produce offspring so that children can inherit much of meaningful information from parents. Using group encoding scheme, chromosomes may have different length. Crossover should be done on chromosomes with varied length.

There are mainly four steps in operator crossover:

(1) Two chromosomes are randomly selected as parents and crossing sites on each parent are chosen at random in both parents.

For example, chromosome $BEA(B\{2,5,7,8\}, E\{1,4\}, A\{9,3,6\})$ and $CBED(C\{5,9\}, B\{2\}, E\{1,6,7\}, D\{4,3,8\})$ are selected as parents. Genes $A\{9,3,6\}$ and $B\{2\}$ are, respectively, crossing sites.

(2) Two parent chromosomes exchange genes at crossing sites.

After exchanging genes, the above two chromosomes become $BEB(B\{2,5,7,8\}, E\{1,4\}, B\{2\})$ and $CAED(C\{5,9\}, A\{9,3,6\}, E\{1,6,7\}, D\{4,3,8\})$.

(3) Some genes with repeated nodes or VMs should be removed. So, the above chromosomes change to $EB(E\{1,4\}, B\{2\})$ and $A(A\{9,3,6\})$.

(4) Some missing VMs are reinserted into genes using first fit decreasing (FFD) heuristic.

In the above example, the missing VMs of the first chromosome include VMs 3, 5, 6, 7, 8, and 9. These VMs should be located on active nodes again. If active nodes do not have enough resource to host these missing VMs, idle nodes are activated.

Crossover operator is done by rate q_c. A population generation produces offsprings with the same size as parents.

Mutation. Mutation may make an individual in the population different from his parents. It adds new information in an arbitrary way to widen search space and avoids being trapped at local optima.

Given a small mutation rate q_m, some chromosomes in the population are selected randomly to execute operator mutation. Mutation is to delete some genes at random in chromosomes. The missing VMs should be relocated to other nodes using FFD.

Selection. Operator selection is to select the new population generation from the old generation and their offsprings. A fast multiobjective genetic algorithm (NSGA-II) [27] is used for operator selection. NSGA-II suits well for constrained multiobjective optimization in any evolutionary algorithm [27].

Each chromosome l has two attributes: nondomination rank (l_{rank}) and crowding distance (l_{distance}) [27]. The smaller the nondomination rank is, the closer the chromosome is to the optimal solution. In the same nondomination rank, the bigger the crowding distance is, the better the chromosome is.

MOGA aims at a resource reconfiguration solution with long stability time of VM placement and small number of VM migrations. Relationship *dominate* between two chromosomes (l, k) is defined as follows:

$$l \textit{ dominate } k, \quad \text{iff } \mathcal{T}_l > \mathcal{T}_k,\ Y_l < Y_k. \tag{8}$$

$\mathcal{T}_l$, $\mathcal{T}_k$ means the stability time of chromosome l, k and Y_l, Y_k denotes the number of VM migration, respectively. Then, we have the following equations (S denotes the set of chromosomes):

$$\begin{aligned} l_{\text{rank}} &= 1, \quad \text{if } \neg\left(\exists k \in S \wedge k \textit{ dominate } l\right). \\ l_{\text{rank}=k_{\text{rank}}+1}, &\quad \text{if } \left(\left(\exists k \in S \wedge k \textit{ dominate } l\right), \text{ for } \left(\forall u \in S \wedge u \textit{ dominate } l\right), \text{ having } k_{\text{rank}} \leq u_{\text{rank}}\right). \end{aligned} \tag{9}$$

The crowding distance is computed as the sum of each normalized objective function [27]. A partial order $\prec$ between two chromosomes l and k is defined. Let $l \prec k$, if ($l_{\text{rank}} < k_{\text{rank}}$ or (($l_{\text{rank}} = k_{\text{rank}}$) and ($l_{\text{distance}} > k_{\text{distance}}$))). Apparently, poset $(S, \prec)$ (S denotes a set of chromosomes in a population generation) is also a well-ordered set. S is a totally ordered set. Chromosomes in set S can be ordered into a chain according to total order $\prec$.

When *popSIZE* parent chromosomes produce *popSIZE* offsprings, all these chromosomes form a big set S' with

($2 * popSIZE$) elements together. Then, selection operator chooses the first $popSIZE$ chromosomes as a new generation from set S' based on total order $\prec$.

4.3. Power-Aware Multiobjective Genetic Algorithm (pMOGA). Power-aware multiobjective genetic algorithm takes power efficiency into consideration based on MOGA. Optimized objectives are listed in formula (3).

Operator crossover and mutation in pMOGA are the same as those in MOGA. Operator selection is discussed below.

Operator selection is still based on NSGA-II. Each chromosome l has two attributes: nondomination rank (l_{rank}) and crowding distance (l_{distance}). The computation of two attributes is like the computation in MOGA. Only crowding distance is computed as the sum of three normalized objective functions in pMOGA, while it is figured out based on two normalized objective functions in MOGA.

In pMOGA, relationship *dominate* between two chromosomes (l, k) is defined in the following:

$$l \text{ dominate } k, \quad \text{iff } \mathcal{T}_l > \mathcal{T}_k,\ Y_l < Y_k,\ A_l < A_k. \tag{10}$$

$\mathcal{T}_l$, $\mathcal{T}_k$ means the stability time of chromosome l, k and Y_l, Y_k denotes the number of VM migration, respectively. Variables A_l and A_k express the number of active physical nodes in chromosome l, k.

4.4. Enhanced Power-Aware Multiobjective Genetic Algorithm (EpMOGA). Enhanced power-aware multiobjective genetic algorithm (EpMOGA) is designed to add a type-matching controller to pMOGA. The controller is mainly used in operator crossover and mutation. EpMOGA and pMOGA have the same operator selection.

As shown in Figures 3 and 4, when placing missing VMs, pMOGA uses FFD and EpMOGA employs a type-matching controller, which is the only difference between pMOGA and EpMOGA.

In cloud computing, the workloads of various applications are multiattribute in terms of different types of resources (CPU, memory, etc.) [28]. A type-matching controller is thus designed to classify applications and nodes into several categories and match them effectively. According to workloads of applications, VMs are classified into CPU-intensive (CI), memory-intensive (MI), both of CPU-intensive and memory-intensive (CMI), none of CPU-intensive and memory-intensive (Non). The type of a VM usually keeps unchanged during their whole lifetime. Also, active physical nodes are sorted into the same four classes. But the type of an active node would vary when it hosts different VMs.

In our experiments, we find that when the same VM migrates to different types of nodes, these nodes have diverse stabilization time. So, we define closeness degree of each type of active nodes for every class of VMs, which is listed in Table 1. As shown in Table 1, the smaller the value of type closeness degree is, the longer the stabilization time of nodes hosting VMs is. When selecting a destination node for a VM, the type-matching controller first tries to match VM to nodes with low type closeness degree. Only when there is not any node with low type closeness degree available are nodes with high closeness degree considered as candidates.

Table 1: Type closeness degree of active nodes to VMs.

Type of VMs	Type of active nodes			
	CI	MI	CMI	Non
CI	4	1	3	2
MI	1	4	3	2
CMI	3	2	4	1
Non	2	3	1	4

When placing missing VMs, type-matching controller tries to map VMs to nodes with appropriate type. It can avoid resource contention and improve resource utilization effectively at the same time to place a CPU-intensive VM on a memory-intensive node. For a CPU-intensive VM, if there is not any memory-intensive active node available, type-matching controller would try to find a node with type Non. If there is not any node with type Non available, a CPU-intensive node is then sought.

5. Performance Evaluation

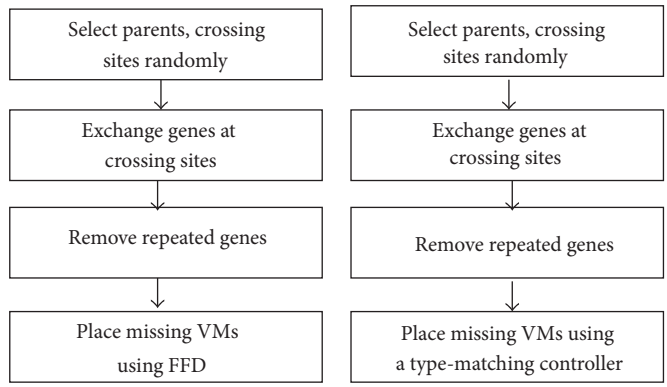
In this section, we evaluate the performance of MOGA, pMOGA, and EpMOGA. All the above algorithms are coded in Java and CloudSim [29] is used to simulate a cloud computing infrastructure. Our tests are done on a ASUS K46CM with Intel Core i5 CPU, 4GB RAM, and 1TB hard drive.

We simulate 58 physical nodes and 174 VMs. Resource requests (only CPU and memory) of these VMs are randomly generated as prediction information. Population size is set as 32 ($popSIZE = 32$). The value of constant MAX_GEN, the maximum generation to produce in genetic algorithms, is set as 40 ($MAX\_GEN = 40$). Crossover rate (q_c) is 0.7 and mutation rate (q_m) is 0.05.

5.1. Evolutionary Process of EpMOGA. Convergence and stability of algorithms are first checked. We observe the evolution process of EpMOGA from the 8th population to the maximum generation.

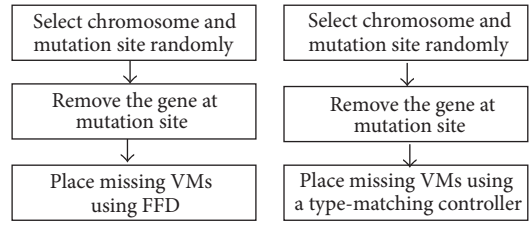
Figure 5 depicts the evolutional process of EpMOGA. x-axis expresses number of VM migrations of each chromosome. y-axis shows stabilization time in seconds. z-axis depicts number of active nodes. Number of VM migrations is just estimated roughly by comparing source node and destination node of each VM. Only five generations (the 8th, 16th, 24th, 32th, and 40th generation) are listed in the figure. Each generation has 32 chromosomes.

From Figure 5, we can find that the reproduction process of individuals moves gradually towards the best solution (longer stabilization time, less number of VM migrations, and less number of active nodes). The process begins with quick changes. The 8th population is quite different from the 16th generation. But the change becomes small in the latter. The 32nd generation is close to the 40th generation. Figure 5 shows that the 40th generation is enough to find the best solution of VM placement in cloud computing.



(a) Crossover in pMOGA (b) Crossover in EpMOGA

Figure 3: Contrast between operator crossovers in pMOGA and EpMOGA.



(a) Mutation in pMOGA (b) Mutation in EpMOGA

Figure 4: Contrast between operator mutations in pMOGA and EpMOGA.

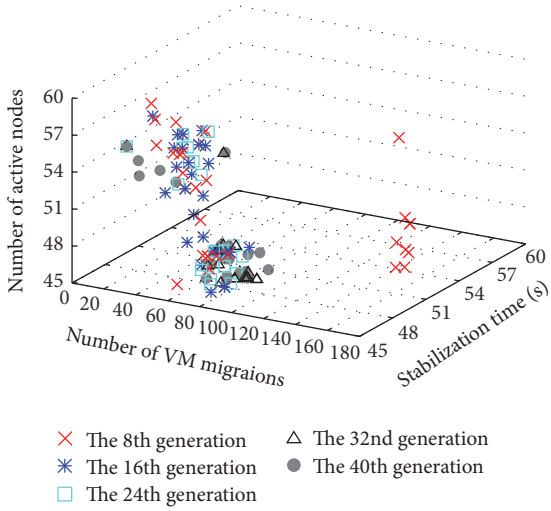


Figure 5: Evolutional process of EpMOGA.

5.2. Comparison of MOGA, pMOGA, and EpMOGA. In this part, we compare the performances of MOGA, pMOGA, and EpMOGA. In environment with the same initial VM placement and the same resource prediction information, MOGA, pMOGA, and EpMOGA, respectively, find a new VM placement. We compare their stabilization time, number of active nodes, and redistribution overhead (denoted as number of VM migrations). Average power $\overline{p}$ is roughly computed using formulae (11).

In formulae (11), $T_{\mathscr{D}}$ denotes stability time of a VM placement pattern $\mathscr{D}$. E_{node} means energy consumed by all active physical nodes ($\sum_{i=1}^{M} y_i$) in pattern $\mathscr{D}$. $\overline{P_{\text{server}}}$ denotes average power of servers. Here, $\overline{P_{\text{server}}}$ is set as 400 watts [7]. E_{mig} expresses energy consumed during VM migration, which is only related to network traffic in migration process [30]. Network traffic is mainly based on the amount of memory of migrated VMs (expressed as $\sum_j Mem'_j$). Parameters k_1, k_2, and k_3 are, respectively, set as 0.512, 1.5, and 20.165, which are got by training models [30].

$$\begin{aligned}\overline{p} &= \frac{\left(E_{\text{node}} + E_{\text{mig}}\right)}{t} \\ &= \frac{\left(T_{\mathscr{D}} * \sum_{i=1}^{M} y_i * \overline{P_{\text{server}}} + \left(k_1 * k_2 * \sum_j Mem'_j + k_3\right)\right)}{T_{\mathscr{D}}}.\end{aligned} \tag{11}$$

We normalize performance values of pMOGA and EpMOGA after setting all the performance values of MOGA as 1. The results are listed in Figure 6. From Figure 6, we find that both pMOGA and EpMOGA have less number of active nodes and less average power at the cost of shorter stability time and larger number of VM migrations. With a type-matching controller, EpMOGA has better performance values than pMOGA. Average power of EpMOGA is 0.818 times that of MOGA and 0.922 times the power of pMOGA.

Figure 6 shows that MOGA has the longest stabilization time and the smallest number of VM migrations. But pMOGA and EpMOGA better balance the relationship

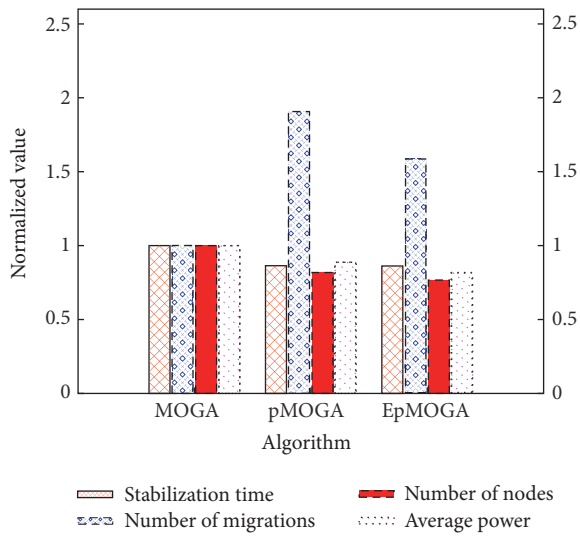


Figure 6: Performance comparison of several algorithms.

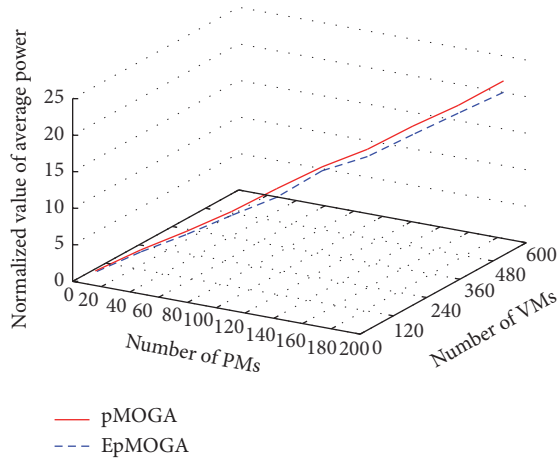


Figure 7: Average power of pMOGA and EpMOGA.

of VM distribution stabilization and power efficiency by adding number of active nodes as one of optimization objectives. Number of active nodes is one of the main power consumption factors in cloud computing. pMOGA and EpMOGA migrate more VMs to use less active nodes, saving more energy consumption. With a type-matching controller, EpMOGA has better solution than pMOGA. The controller helps to optimize evolution process for optimal objectives.

We change number of nodes and number of VMs to test average power of pMOGA and EpMOGA. We set the minimum power in test results as 1 and normalize other power values. The test results are shown in Figure 7. With the increase of VMs and PMs, average power of pMOGA and EpMOGA rises up. EpMOGA always finds a solution with less power than pMOGA. The more VMs and PMs there are, the clearer advantage EpMOGA has. Figure 7 demonstrates that the type-matching controller is helpful to accelerate evolution process for optimal objectives.

6. Conclusion and Future Work

In this paper, several genetic algorithms have been proposed to implement dynamic resource allocation for stability in cloud computing. The group encoding scheme is employed to clearly express the mapping of VMs and physical nodes. A type-matching controller is designed to speed up evolution process. Our simulation results show that these genetic algorithms effectively improve stability of VM redistribution. Also, pMOGA and EpMOGA both better balance the relationship of stabilization and energy efficiency. With type-matching controller, EpMOGA is superior to pMOGA.

In the future, we will continue to work on dynamic resource configuration in cloud computing using genetic algorithms. We find that when there are more objectives to be optimized, nondominated sorting genetic algorithm II is less effective. Many chromosomes are in the same nondomination rank. A new sorting algorithm should be studied.

Abbreviations

$\mathscr{M}$: The total number of physical nodes in the cloud
$\mathscr{N}$: The total number of virtual machines in the cloud
$\mathscr{C}_i$: The amount of CPU resource that node i supplies
Mem_i: The amount of memory resource that node i supplies
$\mathscr{C}'_j$: The amount of CPU resource that VM j requests
Mem'_j: The amount of memory resource that VM j requests
x_{ij}: Binary variable; if $x_{ij} = 1$, node i hosts VM j, or else, $x_{ij} = 0$
y_i: Binary variable; $y_i = 1$ if node i is active and hosts one VM at least, or else, $y_i = 0$
$\mathscr{D}_k$: The kth placement pattern of all VMs in the cloud
m_j: Binary variable; if $m_j = 1$, VM j migrates once, or else, $m_j = 0$
T_{node_i}: Stabilization time of a node, node_i
$T_{\mathscr{D}_k}$: Stabilization time of a placement pattern $\mathscr{D}_k$.

Competing Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This research was supported by the Opening Project of Hubei Key Laboratory of Intelligent Information Processing and Real-Time Industrial System in China (no. 2016znss27B), the National Nature Science Foundation of China (no. 61303117 and no. 61272110), and the Key Projects of National Social Science Foundation of China under Grant no. 11&ZD189.

References

[1] M. Armbrust, A. Fox, R. Griffith et al., "Above the clouds: a Berkeley view of cloud computing," Tech. Rep. UCB/EECS-2009-28, Electrical Engineering and Computer Sciences Department, University of California, Berkeley, 2009.

[2] G. Copil, D. Moldovan, H. Truong, and S. Dustdar, "rSYBL: a framework for specifying and controlling cloud services elasticity," *ACM Transactions on Internet Technology*, vol. 16, no. 3, 2016.

[3] H. Jin, L. Deng, S. Wu, X. H. Shi, H. H. Chen, and X. D. Pan, "MECOM: live migration of virtual machines by adaptively compressing memory pages," *Future Generation Computer Systems*, vol. 38, pp. 23–35, 2014.

[4] A. Rai, R. Bhagwan, and S. Guha, "Generalized resource allocation for the cloud," in *Proceedings of the ACM 3rd Symposium on Cloud Computing (SOCC '12)*, San Jose, Calif, USA, 2012.

[5] F. Hermenier, X. Lorca, J. M. Menaud, G. Muller, and J. Lawall, "Entropy: a consolidation manager for clusters," in *Proceedings of the ACM/Usenix International Conference on Virtual Execution Environments (VEE '09)*, pp. 41–50, Washington, DC, USA, March 2009.

[6] L. Chen and H. Shen, "Consolidating complementary VMs with spatial/temporal-awareness in cloud datacenters," in *Proceedings of the 33rd IEEE Conference on Computer Communications (INFOCOM '14)*, pp. 1033–1041, IEEE, Toronto, Canada, May 2014.

[7] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "An energy-efficient VM prediction and migration framework for overcommitted clouds," *IEEE Transactions on Cloud Computing*, 2016.

[8] L. Zhang, Z. Li, and C. Wu, "Dynamic resource provisioning in cloud computing: a randomized auction approach," in *Proceedings of the 33rd IEEE Conference on Computer Communications ('INFOCOM '14)*, pp. 433–441, Ontario, Canada, May 2014.

[9] Z. Zhou, F. Liu, Z. Li, and H. Jin, "When smart grid meets geo-distributed cloud: an auction approach to datacenter demand response," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM '15)*, pp. 2650–2658, IEEE, May 2015.

[10] L. Deng and L. Yao, "Dynamic allocation of virtual resources based on genetic algorithm in the cloud," in *Proceedings of the Asia-Pacific Services Computing Conference (APSCC '15)*, pp. 153–164, 2015.

[11] S. Nathan, U. Bellur, and P. Kulkarni, "Towards a comprehensive performance model of virtual machine live migration," in *Proceedings of the 6th ACM Symposium on Cloud Computing (SoCC '15)*, pp. 288–301, Kohala Coast, Hawaii, USA, August 2015.

[12] Z.-H. Zhan, X.-F. Liu, Y.-J. Gong, J. Zhang, H. S.-H. Chung, and Y. Li, "Cloud computing resource scheduling and a survey of its evolutionary approaches," *ACM Computing Surveys*, vol. 47, no. 4, article 63, 2015.

[13] T. Kaur and I. Chana, "Energy efficiency techniques in cloud computing: a survey and taxonomy," *ACM Computing Surveys*, vol. 48, no. 2, article 22, 2015.

[14] J. Liu, K. L. Li, D. K. Zhu, J. J. Han, and K. Q. Li, "Minimizing cost of scheduling tasks on heterogeneous multicore embedded systems," *ACM Transactions on Embedded Computing Systems*, vol. 16, no. 2, 2016.

[15] Q. Li, Q.-F. Hao, L.-M. Xiao, and Z.-J. Li, "Adaptive management and multi-objective optimization for virtual machine placement in cloud computing," *Chinese Journal of Computer*, vol. 34, no. 12, pp. 2253–2264, 2011.

[16] W. Wang, B. Li, and B. Liang, "Dominant resource fairness in cloud computing systems with heterogeneous servers," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM '14)*, pp. 583–591, IEEE, Toronto, Canada, May 2014.

[17] J. Guo, F. Liu, J. C. S. Lui, and H. Jin, "Fair network bandwidth allocation in IaaS datacenters via a cooperative game approach," *IEEE/ACM Transactions on Networking*, vol. 24, no. 2, pp. 873–886, 2016.

[18] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Improving resource efficiency at scale with heracles," *ACM Transactions on Computer Systems*, vol. 34, no. 2, 2016.

[19] S. Singh and I. Chana, "QoS-aware autonomic resource management in cloud computing: a systematic review," *ACM Computing Surveys*, vol. 48, no. 3, article 42, 2016.

[20] K. H. Park, W. Hwang, H. Seok et al., "MN-MATE: elastic resource management of manycores and a hybrid memory hierarchy for a cloud node," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 12, no. 1, article 5, 2015.

[21] R. C. Chiang, S. Rajasekaran, N. Zhang, and H. H. Huang, "Swiper: exploiting virtual machine vulnerability in third-party clouds with competition for I/O resources," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 6, pp. 1732–1742, 2015.

[22] N. Jain, I. Menache, J. Naor, and J. Yaniv, "Near-optimal scheduling mechanisms for deadline-sensitive jobs in large computing clusters," *ACM Transactions on Parallel Computing*, vol. 2, no. 1, 2015.

[23] J. Ghaderi, S. Shakkottai, and R. Srikant, "Scheduling storms and streams in the cloud," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, vol. 1, no. 4, 2016.

[24] T. Wu, W. Dou, F. Wu, S. Tang, C. Hu, and J. Chen, "A deployment optimization scheme over multimedia big data for large-scale media streaming application," *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 12, no. 5, article 73, 2016.

[25] J. Xu, C. Liu, X. Zhao, S. Yongchareon, and Z. Ding, "Resource management for business process scheduling in the presence of availability constraints," *ACM Transactions on Management Information Systems*, vol. 7, no. 3, article 9, 2016.

[26] E. Falkenauer and A. Delchambre, "A genetic algorithm for bin packing and line balancing," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1186–1192, Nice, France, May 1992.

[27] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[28] L. Chen, H. Shen, and K. Sapra, "Distributed autonomous virtual resource management in datacenters using finite-Markov decision process," in *Proceedings of the 5th ACM Symposium on Cloud Computing (SOCC '14)*, pp. 1–13, ACM, Seattle, Wash, USA, November 2014.

[29] CloudSim: A Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services, 2015, http://www.cloudbus.org/cloudsim/.

[30] H. Liu, C.-Z. Xu, H. Jin, J. Gong, and X. Liao, "Performance and energy modeling for live migration of virtual machines," in *Proceedings of the 20th ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC '11)*, pp. 171–181, ACM, San Jose, Calif, USA, June 2011.