


# Self-learning and self-adaptive resource allocation for cloud-based software services

Xing Chen<sup>1,2,3</sup> | Junxin Lin<sup>1,2,3</sup>  | Bing Lin<sup>4</sup> | Tao Xiang<sup>1,2,3</sup> | Ying Zhang<sup>5</sup> | Gang Huang<sup>5</sup>

<sup>1</sup>College of Mathematics and Computer Science, Fuzhou University, Fuzhou, China

<sup>2</sup>Key Laboratory of Spatial Data Mining & Information Sharing, Ministry of Education, Fuzhou, China

<sup>3</sup>Fujian Key Laboratory of Network Computing and Intelligent Information Processing, Fuzhou, China

<sup>4</sup>College of Physics and Energy, Fujian Normal University, Fuzhou, China

<sup>5</sup>Key Laboratory of High Confidence Software Technologies, Ministry of Education, Beijing, China

## Correspondence

Gang Huang, Key Laboratory of High Confidence Software Technologies, Ministry of Education, Beijing, China.  
 Email: hg@pku.edu.cn

## Funding information

National Key R&D Program of China, Grant/Award Number: 2017YFB1002000; Natural Science Foundation of China, Grant/Award Number: 61725201 and 61402111; Fujian Collaborative Innovation Center for Big Data Applications in Governments

## Summary

In the presence of scale, dynamism, uncertainty, and elasticity, cloud engineers face several challenges when allocating resources for cloud-based software services. They should allocate appropriate resources in order to guarantee good quality of services as well as low cost of resources. Self-adaptive ability is needed in this process because engineers' intervention is difficult. Traditional self-adaptive resource allocation methods are policy-driven. Thus, cloud engineers usually have to develop separate sets of rules for each systems in order to allocate resources effectively, which leads to high administrative cost and implementation complexity. Machine learning has made great achievements in many fields, and it can be also applied to resource allocation. In this paper, we present a self-learning and self-adaptive approach to resource allocation for cloud-based software services. For a given cloud-based software service, its QoS model is firstly trained on history data, which is capable to predict the QoS value as output by using the information on workload and allocated resources as inputs. Then, on-line decision-making on resource allocation can be carried out automatically based on genetic algorithm, which is aimed to search reasonable resource allocation plan by using the QoS model. We evaluate our approach on RUBiS benchmark, demonstrating the accuracy of the QoS model over 90% and the improvement of resource utilization by 10%-30%.

## KEYWORDS

cloud computing, resource allocation, software adaptation

## 1 | INTRODUCTION

Cloud software engineering paradigm is gaining momentum as evident by the tremendous use of cloud-based software services.<sup>1</sup> To offer scalability and elasticity under changing workloads, cloud providers often have the capability to provide on-demand configurations of software and hardware resources in a shared infrastructure. The elasticity of cloud has caused a paradigm shift in the way we manage cloud-based software services.<sup>2,3</sup> However, by design time, it would be difficult for software engineers and cloud engineers to predict the dynamic changes in workload and the runtime demands of these cloud-based software services. The fact implies that it becomes more complex for engineers to allocate appropriate resources for software services, guaranteeing good quality of services as well as low cost of resources.<sup>4</sup>

Software's ability to adapt at run-time has been proposed as a means to cope with the complexity of today's software systems.<sup>5</sup> Such self-adaptive systems can configure and reconfigure themselves, augment their functionality, continually optimize themselves, protect themselves, and recover themselves, while keeping most of their complexity hidden from the user and administrator.<sup>6</sup> The self-adaptive ability is needed in resource allocation for cloud-based software services, because engineers' intervention is difficult, if not impossible. Traditional self-adaptive resource allocation methods are policy-driven, which are designed by experts.<sup>7,8</sup> However, software services have different characteristics on workload types and resource preferences. So, cloud engineers usually have to develop separate sets of rules for each system in order to allocate resources effectively, which leads to high administrative cost and implementation complexity.

Machine learning that gives computers the ability to learn without being explicitly programmed means it gives system the ability to learn from data,<sup>9</sup> and it has made great achievements in many fields.<sup>10-12</sup> The self-learning ability is needed in self-adaptive resource allocation because self-adaptive management is based on the knowledge, experience, and rules, which are hard to obtain. However, there are still two issues that have to be dealt with.

First, there are large number of indicators and parameters involved with resource allocation. It is difficult to define the expert knowledge of resource allocation and train the suitable model from huge history data based on machine learning.

Second, good quality of services as well as low cost of resources need to be guaranteed when allocating resources for software services. It is hard to make a decision on resource scheduling in an on-line manner, based on expert knowledge.

In this paper, we present a self-learning and self-adaptive approach to resource allocation for cloud-based software services. Our paper makes three major contributions.

- (1) The expert knowledge is defined as the QoS model, which can be trained from huge history data based on machine learning. The QoS model is capable to predict the QoS value as output by using the information on workload and allocated resources as inputs.
- (2) The on-line decision-making on resource allocation can be carried out automatically based on genetic algorithm, which is aimed to search appropriate resource allocation plan by using the QoS model.
- (3) We evaluate our approach on RUBiS benchmark. The evaluation results show that our approach is effective. The accuracy of the QoS model is over 90% and the resource utilization is improved by 10%-30%.

The rest of this paper is organized as follows. Section 2 gives an overview of our approach. Section 3 introduces the establishment of the QoS model. Section 4 presents the on-line decision-making based on genetic algorithm. Section 5 reports the evaluation on RUBiS benchmark. Section 6 discusses related work and we conclude the paper in Section 7.

## 2 | APPROACH OVERVIEW

In this section, we first simulate the problem and then briefly introduce our framework for self-learning and self-adaptive resource allocation.

### 2.1 | Problem statement

The software service has different QoS values when its runtime environment changes. Depending on the nature of initiating agent, environment changes can be classified as external or internal changes. The external changes are initiated by external elements, while the internal changes are applied by the management system. As shown in Table 1, there are three main elements in this problem domain, including external changes, internal changes, and objects. The external changes refer to workloads which have different amounts and types. The internal changes refer to allocated resources, and they consist of several virtual machines which have differences in computing power and price. The objects refer to evaluation values calculated by fitness functions, which make trade-offs between QoS values and resource costs. QoS can include goals for response time, data throughput, and so on, and its value can be calculated by the SLA contract. Resource costs (Cost) mainly come from leased costs ( $Cost_L$ ) and discontinued costs ( $Cost_D$ ) of virtual machines, as shown in Formula (1). Thus, the self-adaptive system is aimed to automatically allocate appropriate resources for software services, guaranteeing good quality of services as well as low cost of resources, according to the workloads.

$$Cost = Cost_L + Cost_D. \quad (1)$$

The expert knowledge is defined as the QoS model that can predict the QoS value as output by using the information on workload and allocated resources as inputs. We train this model from huge history data based on machine learning. Combining objective function, we are able to calculate the evaluation value of software service, given its runtime environment.

Thus, the self-adaptive resource allocation can be transform to the search for most appropriate resource allocation plan. As a classical combinatorial optimization problem, resource-constrained project scheduling problem belongs to NP-Hard puzzle in theory. So, we use genetic algorithm to search most appropriate resource allocation plan based on the fitness function above.

**TABLE 1** Main elements in the problem domain

	Element	Description
External changes	Workloads	The workloads are different in amount and type.
Internal changes	Allocated resources	The allocated resources consist of several virtual machines which are different in computing power and price.
Objects	Evaluation values	The evaluation values are calculated by fitness functions, which make trade-offs between QoS values and resource costs.

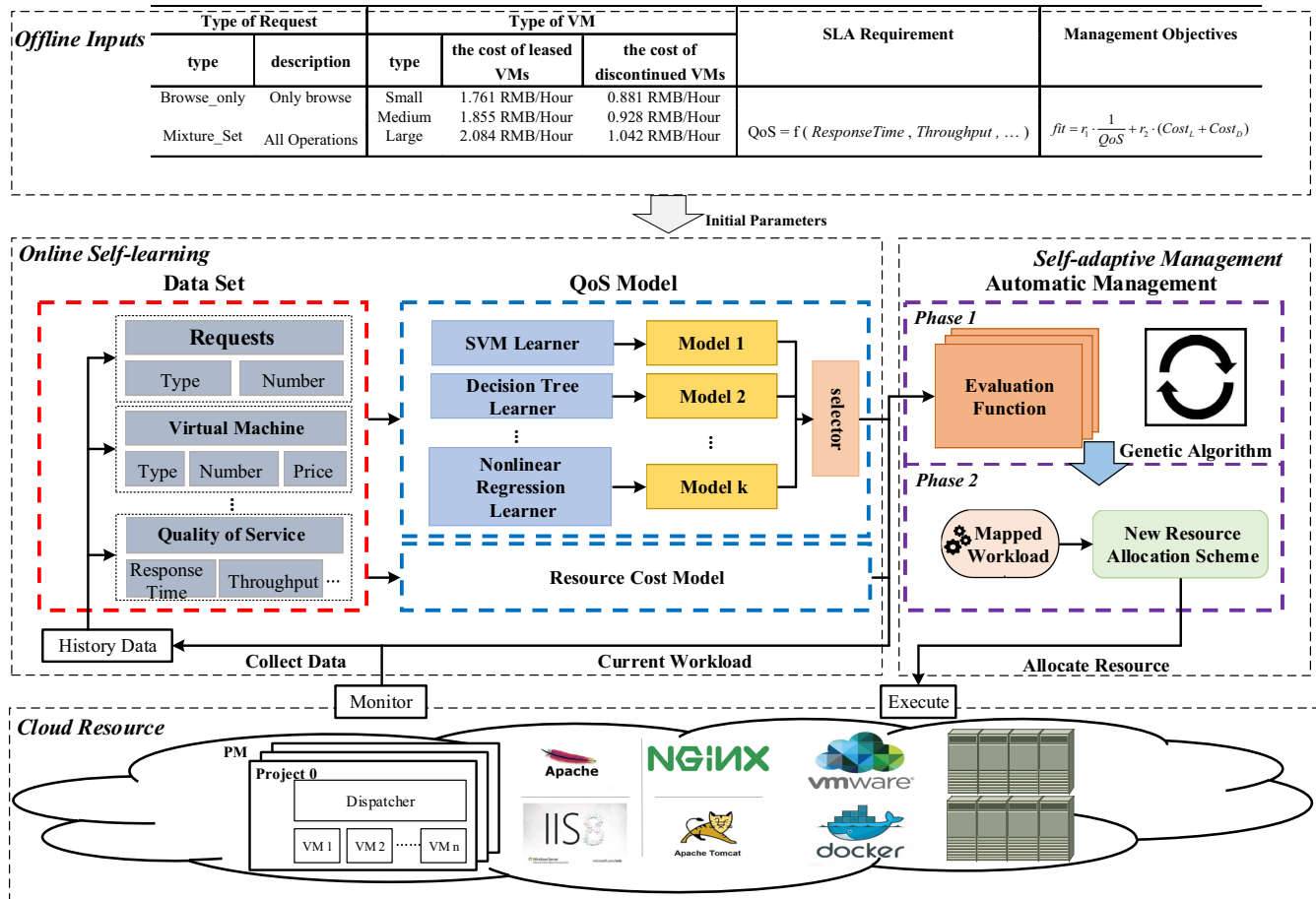


FIGURE 1 Framework for self-learning and self-adaptive resource allocation

## 2.2 | Framework for self-learning and self-adaptive resource allocation

Figure 1 shows our framework for self-learning and self-adaptive resource allocation. There are mainly four modules, including **Cloud Resource** module, **Offline Inputs** module, **Online Self-learning** module, and **Self-adaptive Management** module. **Cloud Resource** module provides APIs to monitor and control cloud resources. **Offline Inputs** module describes basic information about workload, virtual machine, and fitness function. Based on information described in **Offline Inputs** module, **Online Self-learning** module trains the QoS model from history data. Based on the QoS model, **Self-adaptive Management** module uses genetic algorithm to automatically allocate appropriate resources for software services, according to workload changes.

## 3 | QOS MODEL TRAINING

The QoS model is aimed to predict the QoS value as output by using the information on workload and allocated resources as inputs. This section describes how to train the QoS model from history data based on machine learning, which is performed by **Online Self-learning** module.

### 3.1 | Definition of QoS model training

The QoS model is represented as Formula (2). The inputs ( $X$ ) include the amount ( $A$ ) and type ( $T$ ) of workload, and the numbers ( $V$ ) of different types of virtual machines. The output ( $Y$ ) is the predicted value of QoS.

$$Y = Q(X); X = (A, T, V). \quad (2)$$

We use the dataset of history data to train the QoS model, as shown in Table 2. The amount of workload is represented as ( $x_{i,0}$ ). The proportion of different types of tasks in the workload is represented as ( $x_{i,1}, x_{i,2}, \dots, x_{i,m}$ ), and their sum is 100%. The allocated resources are represented as ( $x_{i,m+1}, x_{i,m+2}, \dots, x_{i,m+n}$ ), and  $x_{i,m+s}$  represents the number of virtual machines of  $s^{\text{th}}$  type. The value of QoS is represented as ( $y_{i,0}$ ).

Three machine learning methods, including Nonlinear Regression,<sup>13</sup> Support Vector Machine,<sup>14</sup> and Classification And Regression Tree<sup>15</sup> are used to train the QoS model in this paper. It means that the correlation between input  $X$  and output  $Y$  is explored through the methods above.

TABLE 2 Dataset of history data

A		T				V			Q
$x_{0,0}$	$x_{0,1}$	$x_{0,2}$	$\dots$	$x_{0,m}$	$x_{0,m+1}$	$x_{0,m+2}$	$\dots$	$x_{0,m+n}$	$y_0$
$x_{1,0}$	$x_{1,1}$	$x_{1,2}$	$\dots$	$x_{1,m}$	$x_{1,m+1}$	$x_{1,m+2}$	$\dots$	$x_{1,m+n}$	$y_1$
$x_{2,0}$	$x_{2,1}$	$x_{2,2}$	$\dots$	$x_{2,m}$	$x_{2,m+1}$	$x_{2,m+2}$	$\dots$	$x_{2,m+n}$	$y_2$

## 3.2 | Machine learning methods

### 3.2.1 | Nonlinear regression

For nonlinear regression method, we need to set the regression equation and loss function. The regression equation is shown as Formula (3).

$$y_k = w_0 x_{k,0} + w_1 x_{k,1} + w_2 x_{k,2} + \dots + w_m x_{k,m} + w_{m+1} x_{k,m+1}^2 + w_{m+2} x_{k,m+2}^2 + \dots + w_{m+n} x_{k,m+n}^2 + b. \quad (3)$$

The input matrix  $X$  and output matrix  $Y$  are defined as Formula (2). The matrix  $W$  and parameter  $b$  are required to be solved. In addition, the mean square error is used as loss function, which is minimized through the least squares method.<sup>16</sup>

### 3.2.2 | Support vector machine

For support vector machine (SVM) method, we need to set hyperplane equation and kernel function. The hyperplane equation is shown as Formula (4).

$$Y = u^T \varphi(X) + v. \quad (4)$$

The input matrix  $X$  and output matrix  $Y$  are defined as Formula (2). The parameters  $(u^T, v)$  are solved by transforming into dual problem through Gaussian kernel<sup>17</sup> whose function is shown in Formula (5).

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right), \quad x_i, x_j \in X, \sigma > 0. \quad (5)$$

### 3.2.3 | Classification and regression tree

For classification and regression tree (CART) method, we need to set the calculation equation of dataset purity and Gini index function of attributes.<sup>18</sup> The dataset purity is calculated as Formula (6). The dataset  $D = (X, Y)$  contains the input matrix  $X$  and output matrix  $Y$ , as defined in Formula (2).  $p_k$  is the proportion of the  $k^{\text{th}}$  category in the dataset, and  $r$  is the total categories.

$$\text{Gini}(D) = \sum_{k=1}^{|r|} \sum_{k' \neq k} p_k p_{k'} = 1 - \sum_{k=1}^{|r|} p_k^2 \quad (6)$$

$$\text{Gini}(D, \text{att}) = \sum_{v=1}^r \frac{|D^v|}{|D|} \text{Gini}(D^v). \quad (7)$$

Gini index function of attribute  $\text{att}$  in the attribute column of input matrix  $X$  is described as Formula (7). The one with the smallest Gini index is regarded as the optimal partition attribute.

## 4 | ON-LINE DECISION-MAKING BASED ON GENETIC ALGORITHM

The on-line decision-making on resource allocation is carried out automatically based on genetic algorithm. This section describes how to use genetic algorithm to search the most appropriate resource allocation plan, which is performed by **Self-adaptive Management** module.

### 4.1 | Problem encoding

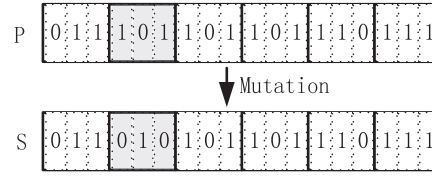
We use binary coding method<sup>19</sup> to encode the resource allocation problem. A chromosome represents a resource allocation plan. We assume that the number of VM types is  $n$ , and the  $j^{\text{th}}$  chromosome in  $t^{\text{th}}$  iteration is defined in Formula (8). The  $k^{\text{th}}$  gene  $x_{ik}^t$  ( $k = 1, 2, \dots, n$ ) represents the number of virtual machines of  $k^{\text{th}}$  type, which is using binary encoding and described as Formula (9).

$$X_j^t = (x_{j1}^t, x_{j2}^t, \dots, x_{jn}^t) \quad (8)$$

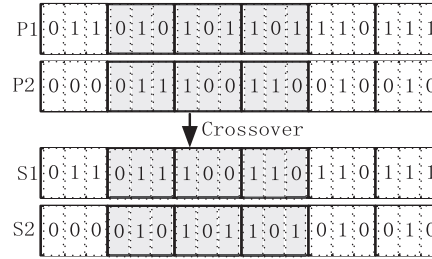
$$x_{ik}^t = \{(0/1)_v, (0/1)_{v-1} \dots (0/1)_1\}_{ik}^t. \quad (9)$$



**FIGURE 2** A chromosome encoded for a resource allocation plan



**FIGURE 3** Mutation operation



**FIGURE 4** Crossover operation

Thus, the value of each gene in a chromosome can range from 0 to  $2^v - 1$ . For instance, Figure 2 shows a chromosome encoded for a resource allocation plan with six VM types in cloud environment. We assume that the maximum number of virtual machines of each type is seven (i.e.  $v = 3$ ), so the value of each gene is in the range of  $[0, 7]$ . The instance describes that the allocated resources include 3  $vm_1$ , 5  $vm_2$ , 5  $vm_3$ , 5  $vm_4$ , 6  $vm_5$ , and 7  $vm_7$ .

## 4.2 | Fitness function

Resource allocation plan is evaluated by the fitness function, which makes trade-offs between QoS values and resource costs, as shown in Formula (10). The fitness function is composed of two parts, namely, the value of QoS and the cost of leased and discontinued virtual machines. Their weights ( $r_1$  and  $r_2$ ) are defined by experts according to requirements for different systems. Also, obviously, resource allocation plans with lower values can be regarded as better ones.

$$fit = r_1 \cdot \frac{1}{QoS} + r_2 \cdot (Cost_L + Cost_D). \quad (10)$$

In addition, the value of QoS can be calculated by the QoS model and the cost of virtual machines can be calculated by the contract. Therefore, given the workload and allocated virtual machines, we can calculate the fitness function value.

## 4.3 | Update strategy

The update operations of genetic algorithm mainly include *selection*, *mutation*, and *crossover*.

**Selection:** Selection operator is used to filter the good chromosomes from the current pool.<sup>20</sup> We select the chromosomes in the current chromosome pool according to the relative probability. The relative probability of a chromosome is defined in Formula (11), which suggests that a chromosome with better fitness value has higher probability to be selected as a new member for the next generation.

$$pro_{sel}(X_i) = 1 - \frac{fit(X_i)}{\sum_{i=1} fit(X_i)}. \quad (11)$$

**Mutation:** Mutation operator is used to generate a chromosome from a single parent.<sup>21</sup> We first randomly select a gene locus in a chromosome. And then, mutate its attribute to a new value (ie, 0 to 1, or 1 to 0). Figure 3 illustrates this process. In Figure 3, gene '101' mutates to '010'.

**Crossover:** Crossover operator is used to breed new chromosomes.<sup>22</sup> A pair of existing chromosomes is regarded as parent chromosomes, which are the operand of crossover operator. The two parent chromosomes exchange a segment of genes to generate the son chromosomes. Figure 4

illustrates this process. 'P1' and 'P2' represent two parent chromosomes. 'S1' and 'S2' represent two son chromosomes. In Figure 4, the genes '010, 101, 101' in 'P1' exchange with the genes '011, 100, 110' in 'P2'.

#### 4.4 | Algorithm flow

The steps of genetic algorithm are briefly described as follows.

**Step 1:** Initializing the values of parameters such as population size, maximum iterations, and initialized population of chromosomes.

**Step 2:** Calculating the fitness values of each chromosomes by Formula (10), and then selecting the chromosome with the lowest fitness function value as the best one.

**Step 3:** Updating the population by *selection*, *mutation*, and *crossover* operations.

**Step 4:** Recalculating the fitness function value of each chromosome, and Updating the best chromosome.

**Step 5:** Going to **Step 3** until the criterion is met.

### 5 | EVALUATION

We evaluate our approach on RUBiS<sup>23</sup> benchmark. The goals of the evaluation are to (1) validate whether the QoS model trained from history data is capable to predict the value of QoS as output by using the information on workload and allocated resources as inputs; (2) compare the performance of on-line decision-making based on genetic algorithm with rule-based approach.

#### 5.1 | Experimental setup

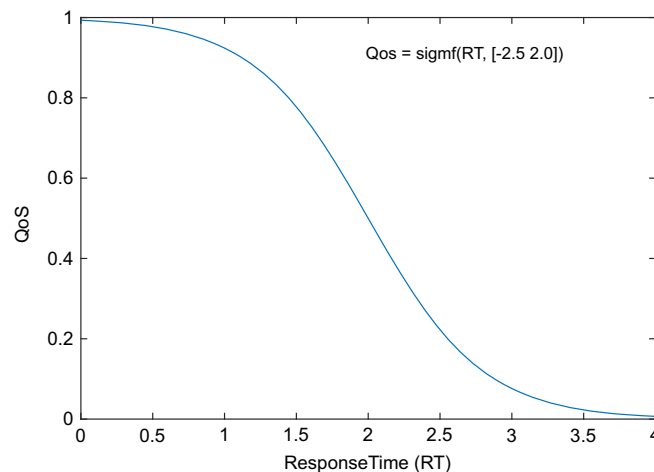
RUBiS is an auction site prototype modeled after eBay.com that is usually used to evaluate application servers performance scalability.<sup>23</sup> It provides a client that emulates users behavior for various workload patterns. And, the number of clients indicates the amount of workload. We assume that the amount of workload is typically in the range of [0, 3000], and there are two types of tasks in the workload. The experiment is carried out in a private cloud and there are three types of virtual machines, as shown in Table 3.

In this experiment, the value of QoS is mainly determined by response time *RT*, and it is mapped into the range of [0, 1], as shown in Figure 5. And, the weights of fitness function and cost model are defined in Formula (12).

$$fit = 10 \cdot \frac{1}{QoS} + 1 \cdot (Cost_L + Cost_D). \quad (12)$$

**TABLE 3** VM types

Property	Small	Medium	Large
CPU	1 core	1 core	1 core
Memory	1 G	2 G	4 G
Price	1.761 RMB/hour	1.885 RMB/hour	2.084 RMB/hour



**FIGURE 5** The QoS contract

## 5.2 | QoS model training

We collected runtime data for two weeks and trained the QoS model based on three machine learning methods respectively. In order to evaluate the accuracy of the QoS model, we define predict value and actual value as  $QoS_{predict}$  and  $QoS_{actual}$ . And, we introduce admissible error  $E$ , confidence interval  $P_r$ , and confidence level  $P$ , as shown in Formula (13).

$$P = P_r (QoS_{actual} - E \leq QoS_{predict} \leq QoS_{actual} + E). \quad (13)$$

We performed 60 sets of tests and the results are shown in Table 4. When the admissible error of QoS is set to 0.1, the confidence levels of QoS models based on Support Vector Machine and Decision Tree are both over 90%. When the admissible error of QoS is set to 0.15, the confidence levels of QoS models based on Support Vector Machine and Decision Tree are both over 95%. From the view of system management, the accuracy of the QoS model is acceptable.

## 5.3 | On-line decision-making

In order to evaluate the on-line decision-making approach based on genetic algorithm, we compare the performance of our approach with rule-based one whose rules are described in Table 5.

We performed two tests in which we emulated workload changes of three hours. As shown in Figure 6, the workload average and the proportion of two tasks are set respectively for each half hour. And, for each half hours, workloads in two tests are set to change in a sinusoidal fashion with different swing values, as shown in Figure 7.

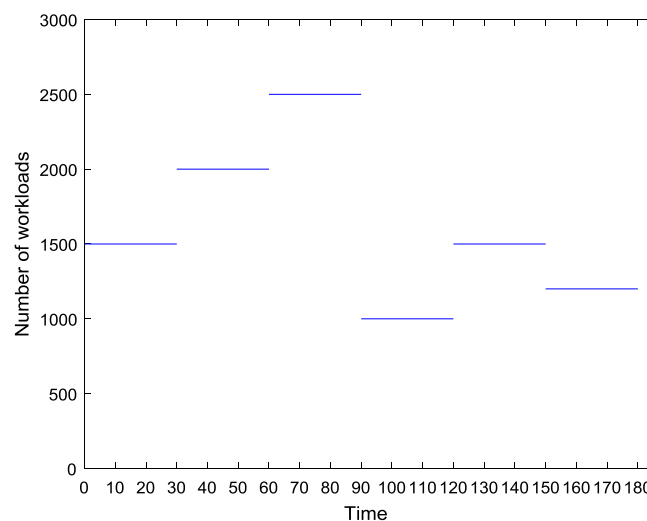
Then, our approach and the rule-based one are used to allocate resources for software services in an automated way, according to workload changes. Their resource allocation plans are illustrated in Table 6 and Table 7. The response times and QoS values of software services are described in Figure 8 and Figure 9. And, the consumption and utilization of CPU and Memory are detailed in Figures 10 and 11. The execution time of genetic algorithm and rule selection is also recorded, as shown in Table 8.

**TABLE 4** Evaluation for QoS models

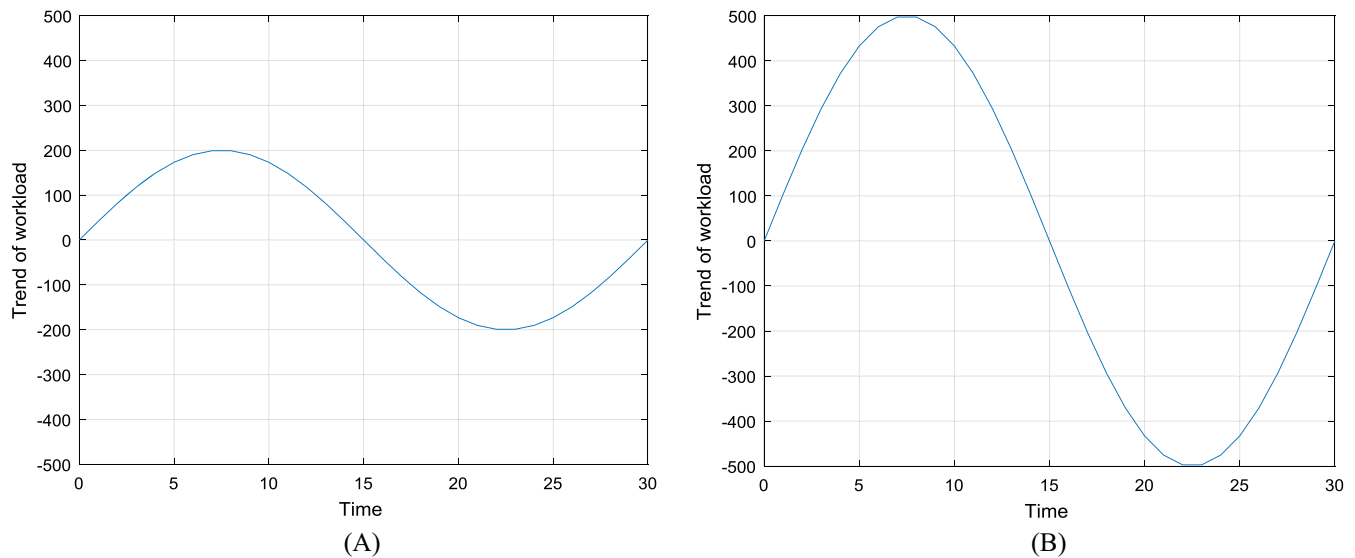
$P$	$E = 0.1$	$E = 0.15$
Nonlinear Regression	83.30%	86.70%
SVM	90.00%	96.70%
CART	91.70%	95.00%

**TABLE 5** The rule-based approach

Condition	Operation
$U_{cpu\_ave} < 30\%$	Shut down any existed VM
$U_{cpu\_ave} > 60\%$	Start a new VM of arbitrary type



**FIGURE 6** The workload average for each half hour



**FIGURE 7** The workload changes in an half hour with different *Swing* values. A, *Swing* = 200; B, *Swing* = 500

**TABLE 6** The resource allocation when *Swing* = 200

Workload average	Allocated VMs of Our Solution				Allocated VMs of Rule-based Solution			
	Small	Medium	Large	Price	Small	Medium	Large	Price
1500	0	0	3		1	2	2	
2000	0	1	3		2	2	2	
2500	0	0	4	45.771	3	2	3	67.0985
1000	0	1	1		2	1	1	
1500	0	0	3		1	2	2	
1200	1	1	1		1	2	1	

**TABLE 7** The resource allocation when *Swing* = 500

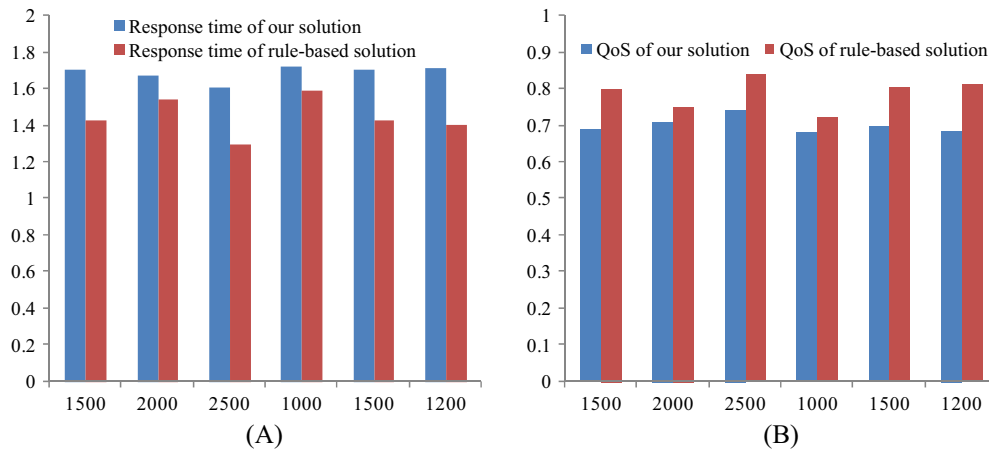
Workload average	Allocated VMs of our solution				Allocated VMs of rule-based solution			
	Small	Medium	Large	Price	Small	Medium	Large	Price
1500	0	1	3		1	2	2	
2000	0	0	4		2	2	2	
2500	0	1	4	55.085	3	2	3	67.0985
1000	1	1	1		2	1	1	
1500	0	1	3		1	2	2	
1200	2	1	1		1	2	1	

We evaluate our approach from three aspects.

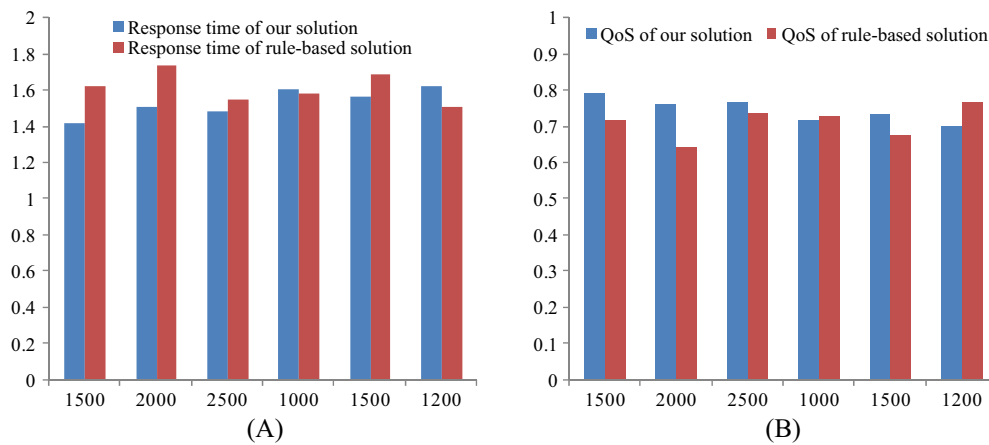
First, we compare QoS values of software services in our solution with the rule-based one. As shown in Figure 8, the QoS values in our solution are slightly lower than the ones in rule-based solution when the workload changes in a sinusoidal fashion with swing value of 200. The average of QoS values in our solution is about 0.7 and the one in rule-based solution is about 0.78. However, as shown in Figure 9, the QoS values in our solution are slightly higher than the ones in rule-based solution when the workload changes in a sinusoidal fashion with swing value of 500. The average of QoS values in our solution is about 0.75 and the one in rule-based solution is about 0.71. So, there is not much difference in QoS values between our solution and rule-based one, and our solution is more stable than the rule-based one. We can see that our approach is capable to maintain QoS values at a reasonable level.

Second, we compare the resource cost of our solution with the rule-based one. As shown in Table 6 and Table 7, the resource cost of our solution is much less than the rule-based one in the precondition of guaranteed QoS. There are two main reasons. On one hand, although the allocated resources of our solution are almost less than the rule-based one, the resources actually used of two solutions are more or less the same, as shown in Figure 10 and Figure 11. It means that the resource utilization of our solution is almost higher than the rule-based one. On the other hand, the three types of virtual machines have different performance/price ratios, and the Large type has the best one in this experiment, as shown in Table 3.





**FIGURE 8** The response times and QoS values of software services when  $Swing = 200$ . A, Response Times; B, QoS Values



**FIGURE 9** The response times and QoS values of software services when  $Swing = 500$ . A, Response Times; B, QoS Values

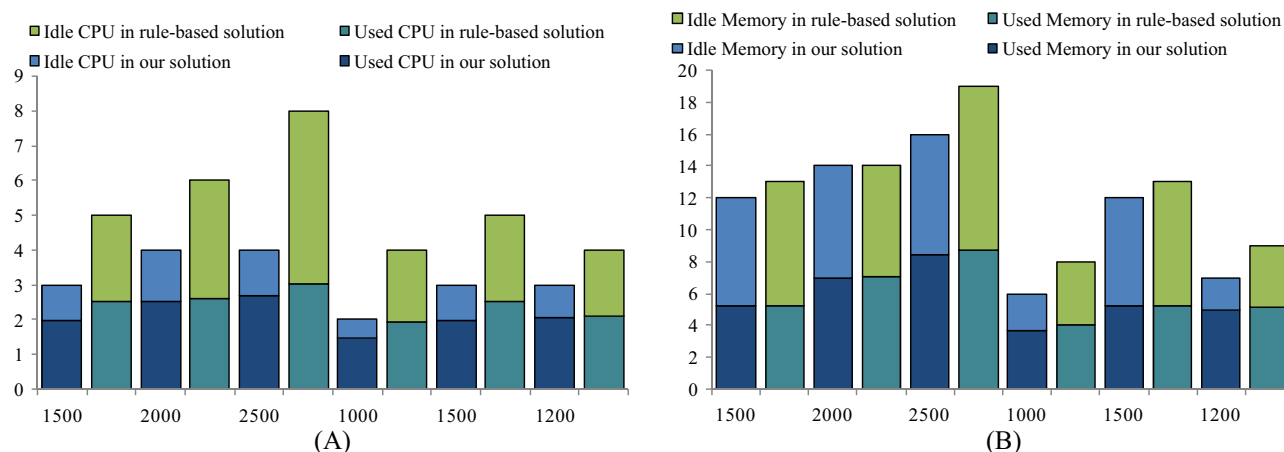
Our approach tends to use the Large type of virtual machine, but the rule-based one use a virtual machine at random from the three types. We can see that our approach is capable to provide the reasonable solution with a better performance/price ratio.

Third, we compare the execution time of our approach with the rule-based one. As shown in Table 8, execution times of our approach are about 12 seconds, which are much more than the rule-based one. However, it can be acceptable from the view of system management.

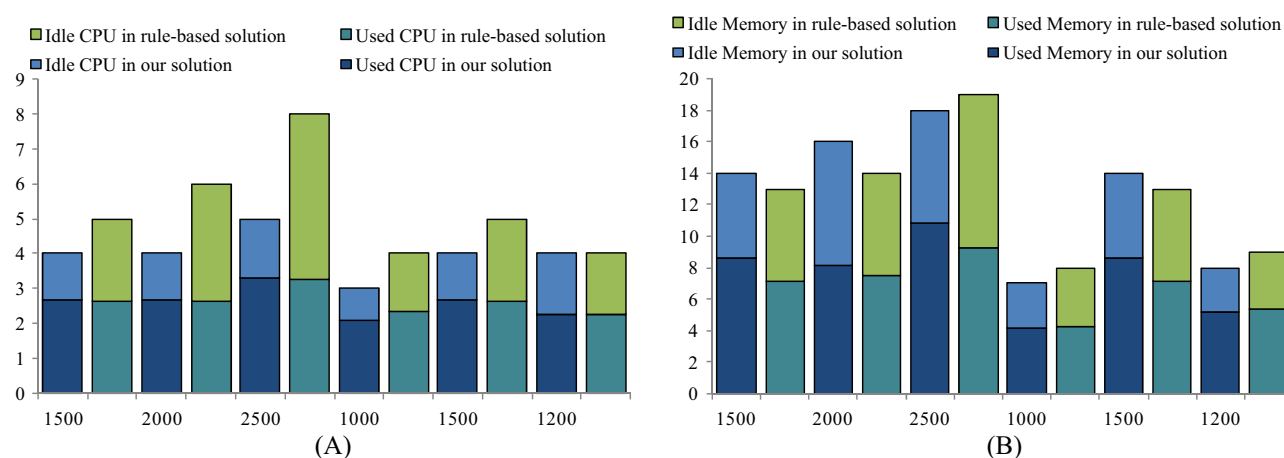
## 6 | RELATED WORK

Cloud engineers need to guarantee good quality of cloud-based software services as well as low cost of resources. In the presence of scale, dynamism, uncertainty, and elasticity, it becomes more complex for engineers to effectively allocate resources for cloud-based software services. The self-adaptive capabilities, including self-configuring,<sup>24</sup> self-optimizing,<sup>25</sup> and self-healing,<sup>26</sup> can cope with the complexity of resource allocation in this scenario.

Traditional self-adaptive resource allocation methods are policy-driven, which are designed by experts.<sup>27–30</sup> Paschke and Bichler<sup>27</sup> propose a rule-based approach in combination with the logical formalisms for SLAs, which specifies rules to trigger after a violation has occurred. Bahati and Bauer<sup>28</sup> introduce an adaptive policy-driven management system which makes use of reinforcement learning methodologies to determine how to best use a set of active policies to meet different performance objectives. Maurer et al<sup>29</sup> present a dynamic resource configuration to achieve high resource utilization and low service level agreement violation rates using knowledge management techniques (Case-Based Reasoning and a rule-based approach). Jamshidi et al<sup>30</sup> exploit fuzzy logic to enable qualitative specification of elasticity rules for cloud-based software. However, there are differences in workload type and resource preference between software services. So, cloud engineers usually have to develop separate sets of rules for each system, which leads to high administrative cost and implementation complexity.



**FIGURE 10** The consumption and utilization of CPU and memory when  $Swing = 200$ . A, CPU Consumption and Utilization; B, Memory Consumption and Utilization



**FIGURE 11** The consumption and utilization of CPU and memory when  $Swing = 500$ . A, CPU Consumption and Utilization; B, Memory Consumption and Utilization

**TABLE 8** Execution times of genetic algorithm and rule selection

Workload	$swing = 200$		$swing = 500$	
	Genetic algorithm	Rule selection	Genetic algorithm	Rule selection
1500	12.488	1.023	13.848	0.983
2000	15.47	0.874	14.4	0.906
2500	11.927	1.218	14.221	1.172
1000	11.978	1.147	12.553	1.134
1500	11.979	1.209	11.072	1.186
1200	14.87	0.964	11.224	1.02

Some recent works also introduce machine learning to self-adaptive resource allocation.<sup>31–33</sup> Farahnakian et al<sup>31</sup> propose a reinforcement-learning-based mechanism to dynamically adjust CPU and memory thresholds for each physical machine. Yan et al<sup>32</sup> present a reinforcement-learning-based approach to learning the results of application behaviors and changing its resource allocation plans. Zheng et al<sup>33</sup> use a load predictor that clusters historical resource utilization and select the cluster set with the highest similarity as a training sample into a neural network. All the works train the resource-management model that directly predicts the management scheme as output by using the information on application runtime environment as inputs. And, they could be just applied to some simple cases because, in complex cases, there are so huge system states that will lead to large search spaces when training direct prediction model.

Our framework can automatically allocate resources for cloud-based software services based on genetic algorithm by using the QoS model that is trained from history data. It is different from works above mainly in two aspects. First, our framework divides the solution of resource allocation

into two parts, including the QoS model and the decision algorithm, which reduces greatly the search space when training prediction model. Second, our framework can be used in most cases of resource allocation because the QoS model and the fitness function can be tuned according to different requirements. In addition, our framework is independent from workload changes, that is, predicts workload and allocates resources independently. Therefore, our framework is capable to interwork with workload prediction models.<sup>34</sup>

## 7 | CONCLUSION AND FUTURE WORK

This paper presents a self-learning and self-adaptive approach to resource allocation for cloud-based software services. First, the QoS model can be trained from huge history data based on machine learning, which is capable to predict the QoS value as output by using the information on workload and allocated resources as inputs. Second, the on-line decision-making on resource allocation can be carried out automatically based on genetic algorithm, which is aimed to search appropriate resource allocation plan by using the QoS model. We evaluate our approach on RUBiS benchmark, demonstrating the accuracy of the QoS model over 90% and the improvement of resource utilization by 10%-30%.

Our approach will be validated and further improved as follows. On one hand, more experiments with real-world software services will be done, especially on public cloud. On the other hand, the feedback mechanism will be introduced to the on-line decision-making process in order to improve its management effects.

## ACKNOWLEDGMENTS

This work was supported by the National Key R&D Program of China under grant 2017YFB1002000, the Natural Science Foundation of China under grants 61725201 and 61402111, and the Fujian Collaborative Innovation Center for Big Data Applications in Governments.

## ORCID

Junxin Lin  <http://orcid.org/0000-0002-0773-0167>

## REFERENCES

1. Gupta A, Kale LV, Gioachin F, et al. The who, what, why, and how of high performance computing in the cloud. Paper presented at: 2013 IEEE 5th International Conference on Cloud Computing Technology and Science; 2013; Bristol, UK.
2. Saa P, Moscoso-Zea O, Costales AC, Luján-Mora S. Data security issues in cloud-based software-as-a-service ERP. Paper presented at: 2017 12th Iberian Conference on Information Systems and Technologies (CISTI); 2017; Lisbon, Portugal.
3. Meena M, Bharadi VA. Performance analysis of cloud based software as a service (SaaS) model on public and hybrid cloud. Paper presented at: 2016 Symposium on Colossal Data Analysis and Networking (CDAN); 2016; Indore, India.
4. Lin K, Dumitrescu S. Cross-layer resource allocation for scalable video over OFDMA wireless networks: tradeoff between quality fairness and efficiency. *IEEE Trans Multimed*. 2017;19(7):1654-1669.
5. Chen T, Bahsoon R. Self-adaptive and online QoS Modeling for cloud-based software services. *IEEE Trans Softw Eng*. 2017;43(5):453-475.
6. Zhou Y, Han J, He J, Zhang H, Xiao Z. SADLBolB: Self-adaptive dynamic load balance over InfiniBand. Paper presented at: 2006 Japan-China Joint Workshop on Frontier of Computer Science and Technology; 2006; Fukushima, Japan.
7. Yu L, Jin H, Jiang W, Liao G, Liao X. Self-adaptive schedule mechanism for peer-to-peer multi-rate live streaming system. Paper presented at: 2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems; 2012; Liverpool, UK.
8. Horst J, Noble J. Distributed and centralized task allocation: When and where to use them. Paper presented at: 2010 4th IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshop; 2010; Budapest, Hungary.
9. Nakib A, Hilia M, Heliore F, Talbi EG. Design of metaheuristic based on machine learning: A unified approach. Paper presented at: 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW); 2017; Lake Buena Vista, FL.
10. Shakoort MT, Rahman K, Rayta SN, Chakrabarty A. Agricultural production output prediction using supervised machine learning techniques. Paper presented at: 2017 1st International Conference on Next Generation Computing Applications (NextComp); 2017; Mauritius, Mauritius.
11. Nojima Y, Arahari K, Takemura S, Ishibuchi H. Multiobjective fuzzy genetics-based machine learning based on MOEA/D with its modifications. Paper presented at: 2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE); 2017; Naples, Italy.
12. Costello JJA, West GM, McArthur SDJ. Machine learning model for event-based prognostics in gas circulator condition monitoring. *IEEE Trans Reliab*. 2015;66(4):1048-1057.
13. Papageorgiou G, Bouboulis P, Theodoridis S. Robust nonlinear regression: a greedy approach employing kernels with application to image denoising. *IEEE Trans Signal Process*. 2017;65(16):4309-4323.
14. Zhang T, Chen W. LMD based features for the automatic seizure detection of EEG signals using SVM. *IEEE Trans Neural Syst Rehab Eng*. 2017;25(8):1100-1108.
15. Zhong S, Tam KS. Hierarchical classification of load profiles based on their characteristic attributes in frequency domain. *IEEE Trans Power Syst*. 2015;30(5):2434-2441.

16. Dao TV, Chaitusaney S, Nguyen HTN. Linear least-squares method for conservation voltage reduction in distribution systems with photovoltaic inverters. *IEEE Trans Smart Grid*. 2017;8(3):1252-1263.
17. Kumar P, Biswas M. SVM with Gaussian kernel-based image spam detection on textual features. Paper presented at: 2017 3rd International Conference on Computational Intelligence & Communication Technology (CICT); 2017; Ghaziabad, India.
18. Loza A, Principe JC. Transient model of EEG using Gini index-based matching pursuit. Paper presented at: 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP); 2016; Shanghai, China.
19. Jodavi M, et al. JSObfusDetector: A binary PSO-based one-class classifier ensemble to detect obfuscated JavaScript code. Paper presented at: 2015 The International Symposium on Artificial Intelligence and Signal Processing (AISP); 2015; Mashhad, Iran.
20. Mistry K, Zhang L, Neoh SC, Lim CP, Fielding B. A micro-GA embedded PSO feature selection approach to intelligent facial emotion recognition. *IEEE Trans Cybern*. 2017;47(6):1496-1509.
21. Duzanec, Kovacic Z. Determination of optimal mutation interval for  $\mu$ ga leased on the performance analysis of GA and  $\mu$ GA. Paper presented at: 2009 European Control Conference (ECC); 2009; Budapest, Hungary.
22. Roy P. A new memetic algorithm with GA crossover technique to solve single source shortest path (SSSP) problem. Paper presented at: 2014 Annual IEEE India Conference (INDICON); 2014; Pune, India.
23. Quemard JP. RUBIS: from civilian technologies to military applications. Paper presented at: Military Communications Conference, 1993 Communications on the Move, IEEE; 1993; Boston, MA.
24. Cirani S, Davoli L, Ferrari G, et al. A scalable and self-configuring architecture for service discovery in the internet of things. *IEEE Internet Things J*. 2014;1(5):508-521.
25. Pederzolli F, Siracusa D, Salvadori E, Cigno RL. Energy saving through traffic profiling in self-optimizing optical networks. *IEEE Syst J*. 2017;11(2):752-761.
26. Leite JB, Mantovani JRS. Development of a self-healing strategy with multiagent systems for distribution networks. *IEEE Trans Smart Grid*. 2017;8(5):2198-2206.
27. Paschke A, Bichler M. Knowledge representation concepts for automated SLA management. *Decis Support Syst*. 2009;46(1):187-205.
28. Bahati RM, Bauer MA. Adapting to run-time changes in policies driving autonomic management. Paper presented at: 4th International Conference on Autonomic and Autonomous Systems (ICAS'08); 2008; Gosier, Guadeloupe.
29. Maurer M, Brandic I, Sakellariou R. Adaptive resource configuration for cloud infrastructure management. *Futur Gener Comput Syst*. 2013;29(2):472-487.
30. Jamshidi P, Ahmad A, Pahl C. Autonomic resource provisioning for cloud-based software. Paper presented at: Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems; 2014; Hyderabad, India.
31. Farahnakian F, Pahikkala T, Liljeberg P, Plosila J, Tenhunen H. Utilization prediction aware VM consolidation approach for green cloud computing. Paper presented at: 2015 IEEE 8th International Conference on Cloud Computing; 2015; New York, NY.
32. Yan Y, Zhang B, Guo J. An adaptive decision making approach based on reinforcement learning for self-managed cloud applications. Paper presented at: 2016 IEEE International Conference on Web Services (ICWS); 2016; San Francisco, CA.
33. Zheng S, Zhu GB, Zhang J, Feng W. Towards an adaptive human-centric computing resource management framework based on resource prediction and multi-objective genetic algorithm. *Multimed Tools Appl*. 2015;76(17):17821-17838.
34. Zhao T, Zan T, Zhao H, Hu Z, Jin Z. Integrating goal model into rule-based adaptation. Paper presented at: 2016 23rd Asia-Pacific Software Engineering Conference (APSEC); 2016; Hamilton, New Zealand.

**How to cite this article:** Chen X, Lin J, Lin B, Xiang T, Zhang Y, Huang G. Self-learning and self-adaptive resource allocation for cloud-based software services. *Concurrency Computat Pract Exper*. 2019;31:e4463. <https://doi.org/10.1002/cpe.4463>