Sui et al. EURASIP Journal on Wireless Communications and Networking (2019) 2019:160
https://doi.org/10.1186/s13638-019-1454-9

EURASIP Journal on Wireless Communications and Networking

RESEARCH Open Access

Virtual machine scheduling strategy based on machine learning algorithms for load balancing

Xin Sui[1,2], Dan Liu[1], Li Li[1*], Huan Wang[1] and Hongwei Yang[1]

Abstract

With the rapid increase of user access, load balancing in cloud data center has become an important factor affecting cluster stability. From the point of view of green scheduling, this paper proposed a virtual machine intelligent scheduling strategy based on machine learning algorithm to achieve load balancing of cloud data center. Firstly, a load forecasting algorithm based on genetic algorithm (SVR_GA), *k*-means clustering algorithm based on optimized min-max, and adaptive differential evolution algorithm (ESA_DE) to enhance local search ability are proposed to solve the load imbalance problem in cloud data center. The experimental results showed that compared with other classical algorithms, the proposed virtual machine scheduling strategy reduces the number of virtual machine migration by 94.5% and the energy consumption of cloud data center by 49.13%.

Keywords: Load balancing, Virtual machine, Support vector regression, Clustering, Differential evolution

1 Introduction

Load balancing plays an important role in resource management of cloud data center. It can not only improve the resource utilization of servers and prevent servers from being overloaded, but also effectively reduce the migration frequency of virtual machines and avoid unnecessary waste of resources. At present, most load balancing strategies are optimized from the following aspects: load balancing strategy based on server CPU and memory resource utilization, load balancing migration strategy based on service-level agreement (SLA), load balancing strategy based on network traffic prediction, load balancing strategy based on quality of service (QoS), load balancing strategy based on service response time prediction, and load balancing strategy based on cloud storage.

Yu et al. proposed a stochastic load balancing scheme which aimed to provide probabilistic guarantee against the resource overloading with virtual machine migration, while minimizing the total migration overhead. Our scheme effectively addressed the prediction of the distribution of resource demand and the multidimensional resource requirements with stochastic characterization. Moreover, as opposed to the previous works that measured the migration cost without considering the network topology, our scheme explicitly took into account the distance between the source physical machine and the destination physical machine for a virtual machine migration. The trace-driven experiments showed that our scheme outperformed the previous schemes in terms of SLA violation and the migration cost [1].

Elrotub and Gherbi adopted the machine learning technique, which was classification, was used to make groups of VMs based on their CPU and RAM utilization, as well as to classify user jobs/tasks into different groups based on their sizes and based on information from log files. The approach arranged virtual machines in groups, and several tasks shared the same VM resources. The goal of our proposal was to allow more dynamic resources and to improve the QoS requirements by maximizing the usage of the resources and user satisfaction, such as increasing resource utilization and reducing the number of job rejections [2].

Ramezani developed a multi-objective load balancing (MO-LB) system that avoided VM migration and

* Correspondence: ll@cust.edu.cn
[1]College of Computer Science and Technology, Changchun University of Science and Technology, Changchun 130022, China
Full list of author information is available at the end of the article

achieved system load balancing by transferring extra workload from a set of VMs allocated on an overloaded PM to other compatible VMs in the cluster with greater capacity. To reduce the time factor even more and optimize load balancing over a cloud cluster, MO-LB contained a CPU usage prediction (CUP) sub-system. The CUP not only predicted the performance of the VMs but also determined a set of appropriate VMs with the potential to execute the extra workload imposed on the VMs of an overloaded PM [3].

Hieu et al. presented a virtual machine consolidation algorithm with usage prediction (VMCUP) for improving the energy efficiency of cloud data centers. Their algorithm was executed during the virtual machine consolidation process to estimate the short-term future CPU utilization based on the local history of the considered servers. The joint use of the current and predicted CPU utilization metrics allowed a reliable characterization of overloaded and underloaded servers, thereby reducing both the load and the power consumption after consolidation [4].

Chen et al. proposed a load balancing algorithm based on server running state, which could calculate comprehensive loading according to the CPU utilization, memory utilization, and network traffic of the servers. Furthermore, a load balancing solution based on software-defined networks technology (SDN) was applied in this paper, and it was designed and implemented in OpenFlow network. We combined network management and server state monitor in this scheme, in which the OpenFlow switched forward the request to the least comprehensive loading server by modifying the packet [5].

In cloud computing, load balancing is a technique to distribute the workload for balancing between two or more cloud servers. Load balancing aims to optimize resource use, maintain the cost of data center and virtual machines, maximize throughput, minimize response time, and avoid overload of any single resource. The main objective of this research paper was to reduce cost and response times using throttled load balancing policy across VM's in multi data center and optimize response time service broker policy. This study has evaluated the throttled load balancing algorithm and their scheduling criteria like overall response time, data center processing time, and total cost of virtual machine and data transfer cost [6].

From the perspective of green scheduling, this paper studied the load balancing strategy of cloud data center from three aspects: server load forecasting, virtual machine selection, and target server selection. The main work was as follows:

1. Aiming at the optimization problem of three parameters ε, C, and σ in the support vector regression algorithm, a load forecasting algorithm based on support vector regression optimized by genetic algorithm was proposed, which could accurately predict the CPU utilization of servers.
2. Aiming at the problem of the cluster number and the selection of initial cluster centers in *k*-means algorithm, this paper proposed a *k*-means clustering algorithm based on optimized min-max. This algorithm found virtual machines with less migration cost, network traffic, and performance interference from overloaded servers.
3. In this paper, the traditional differential evolution algorithm was improved, and an adaptive differential evolution algorithm (ESA_DE) was proposed to enhance the local search ability. The virtual machine found by clustering algorithm was migrated to the target server with the lowest migration cost, network traffic, and performance disturbance, so as to achieve load balancing of cloud data center.

2 Support vector regression load prediction based on genetic algorithms

When the tasks on the virtual machine executed a period of time, some tasks have been completed, and the virtual machine where these tasks were located would release the server resources, which might cause the server to be in a low-load state, while some tasks required a lot of computation, and the virtual machine where these tasks were located would occupy as much server resources as possible, resulting in the server being overloaded. In order to balance the load of cloud data centers, it was necessary to predict the load status of servers and determine whether the servers were overloaded or under-loaded in the next step. In order to accurately predict the load state of servers, this paper studied the traditional algorithm of support vector regression, found out three parameters to solve the accuracy of the traditional support vector regression algorithm, and proposed a load prediction algorithm based on genetic algorithm optimization support vector regression (SVR_GA).

In cloud environment, the load state of servers was related to the number of virtual machines, resource requests for each virtual machine, virtual machine migration, the memory size of the migrated virtual machine, network flow, and the processing capacity of the current server. The load state of the server was non-linear, sudden, and periodic to some extent. Therefore, in the process of solving support vector regression, the Gauss kernel was studied as a kernel function.

The insensitive loss function of support vector regression (SVR) could be expressed as formula (1), in which

the parameters ε represented the approximation accuracy of training data points, that was, the maximum allowable deviation.

$$L_\varepsilon(f(x_i)-y_i) = \left\{ \begin{array}{c} 0, \mid f(x_i)-y_i \mid \leq \varepsilon \\ \mid f(x_i)-y_i \mid -\varepsilon, \mid f(x_i)-y_i \mid > \varepsilon \end{array} \right\} \quad (1)$$

Therefore, the support vector regression SVR model could be simplified to formula (2), in which parameter C played a balancing role in the model complexity and training error.

$$\min_{\omega,b} \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{m} L_\varepsilon(f(x_i)-y_i) \quad (2)$$

The Gauss kernel function was shown in formula (3), in which the parameter σ represented the approximation accuracy of the Gauss kernel function [7].

$$k(x_i, x_j) = \exp\left(-\frac{\|x_i-x_j\|^2}{2\sigma^2}\right) \quad (3)$$

These three parameters usually needed to be set manually by users. In order to obtain the optimal prediction results, this paper used genetic algorithm to optimize the three parameters ε, C, and σ in the process of solving the support vector regression model. The flow chart is shown in Fig. 1.

The load forecasting algorithm based on genetic algorithm optimized support vector regression (SVR_GA) is as follows:

Algorithm1 SVR_GA

input: trainData, predictData
input: Three parameters ε, C, σ, max iteration times
output: predictResult
1. start
2. trainDataNormal = Normalization(trainData);// Normalization of training data
3. predictDataNormal = Normalization(predictData);// Normalization of Prediction Data
4. iterationIndex = 0;
5. minError = MAX;
6. $population_{set}$ = Init(ε, C, σ);// Initialization of population set
7. while(iterationIndex<maxIterationTimes)
8. // Initialization parameters of genetic algorithm
9. $\varepsilon_{optimization}, C_{optimization}, \sigma_{optimization} = GeneticAlgorithm(population_{set})$
10. // Optimized Parameter Generation Support Vector Regression Gauss Model
11. $svr = SVR(kernel = 'rbf', C = C_{optimization}, gamma = \sigma_{optimization}, epsilon = \varepsilon_{optimization})$
12. svr.fit(trainDataNormal.x, trainDataNormal.y); //support vector regression model training data
13. y = svr.predict(trainDataNormal.x); //support vector regression verification training data
14. if(abs(y - trainDataNormal.y) <minError)
15. minError = abs(y - trainDataNormal.y)
16. $\varepsilon_{best}, C_{best}, \sigma_{best} = \varepsilon_{optimization}, C_{optimization}, \sigma_{optimization}$
17. endif
18. iterationIndex ++;
19. endwhile
20. svr=SVR(kernel='rbf',C= C_{best}, gamma= σ_{best}, epsilon= ε_{best});
21. predictResult = svr.predict(predictDataNormal.x);
22. return predictResult;
23. end

3 *k*-means clustering algorithm-based optimized min-max (K-Means-OMM)

The traditional *k*-means clustering algorithm had the following problems: the selection of *k* value in the *k*-means clustering algorithm was very difficult to estimate, and it was not known in advance how many categories the samples should be divided into. Clustering algorithm was very sensitive to the initial clustering mean, and different initial clustering mean would lead to different clustering results. If the distance of the initial clustering mean was very close, the number of iterations and the time of clustering would be greatly increased in the process of clustering, and it was easy to fall into the local optimal solution. During the implementation of the clustering algorithm, it was necessary to constantly calculate the mean of each cluster and adjust the sample classification according to the mean of each cluster. When the number of sample sets was very large, the execution time of the algorithm was long. If there were outliers in the sample set, the clustering mean would deviate seriously.

Aiming at the problems of the *k*-means clustering algorithm, this paper optimized the *k*-means algorithm from two aspects of clustering number *k* and initial clustering center and proposed an optimized min-max *k*-means algorithm (K-Means-OMM) for clustering number and initial clustering center.

3.1 Selecting *k* value of cluster

In the *k*-means clustering algorithm, selecting *K* value was very important for clustering results. This paper classified virtual machines from three aspects: migration cost, server load, and performance interference. The clustering results of virtual machines could be displayed as three-dimensional cubes. Migration cost could be divided into two categories: high migration cost and low migration cost. Performance interference could be divided into two categories: high-performance interference and low-performance interference. Server load state could be divided into three states: high load, low load, and normal load. Therefore, this paper set *K* value of 12; the specific description of the category could be seen in Table 1.

3.2 Selection of initial clustering centers

The *k*-means clustering algorithm usually chose *K* samples randomly as the initial clustering centers and completed the clustering process by iteration. Different initial clustering centers would produce different clustering results. The sensitivity of the *k*-means algorithm to the initial clustering centers would lead to the instability of clustering results. The results of each clustering were different and not the optimal solution.

In this paper, the selection of initial clustering centers of the *k*-means algorithm was optimized to maximize the distance between the initial clustering centers, which could reduce the number of iterations and prevent the

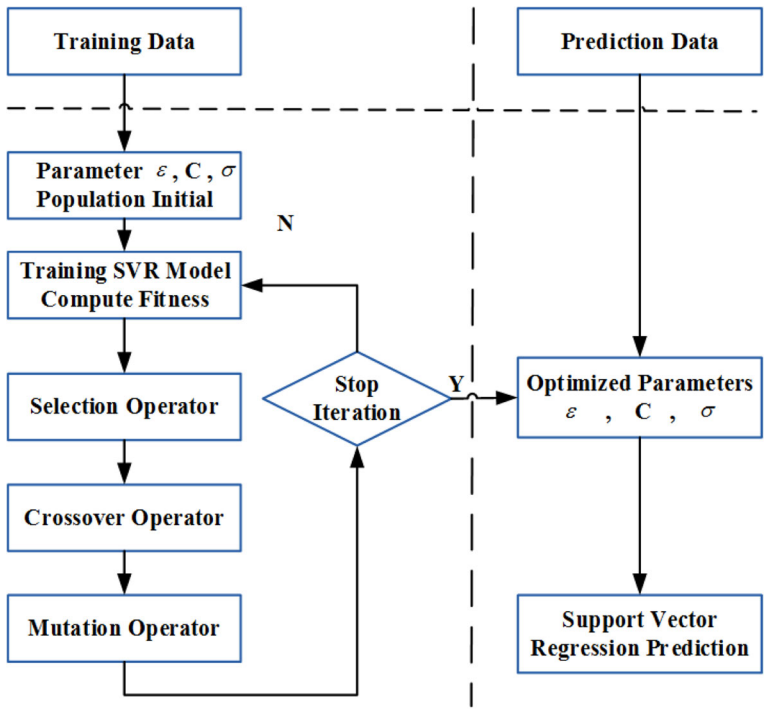

Fig. 1 Flow chart of support vector regression optimized by genetic algorithms

occurrence of local optimal solutions. In this paper, three clustering center selection schemes were studied experimentally.

The first scheme: randomly selected K samples as initial clustering centers and increased the number of iterations in the clustering process. This method was simple but ineffective.

The second scheme: using the min-max algorithm to select the initial clustering centers. Firstly, a sample was randomly selected as the first clustering center C_0; secondly, calculating the distance between each sample point and C_0, the farthest sample point served as the second clustering center C_1; calculate the distance between the rest of the sample points and the cluster center C_0 and C_1 respectively; choose the smaller distance, and then choose the largest distance from these distance as the third clustering center. By analogy, K clustering centers were finally found [8].

Table 1 Virtual machine classification

Serial number	Migration cost	Performance interference	Load state
1	High	High	High
2	High	High	Low
3	High	High	Normal
4	High	Low	High
5	High	Low	Low
6	High	Low	Normal
7	Low	High	High
8	Low	High	Low
9	Low	High	Normal
10	Low	Low	High
11	Low	Low	Low
12	Low	Low	Normal

The third scheme: this paper optimized it on the basis of the min-max algorithm. Firstly, two sample points farthest from each sample point were calculated as the first and second clustering centers C_0 and C_1. Secondly, the distances between each sample point and the clustering centers C_0 and C_1 were calculated separately. The smaller distance was selected to form a set, and the largest distance was selected as the third clustering center from the smaller set, and K clustering centers were selected by analogy.

As shown in Algorithm 2, lines 3–10: find the two farthest sample points as C_0 and C_1; lines 11–13: calculate the sample points and the initial clustering center C_0 , C_1, …, C_j; line 15: choose the shortest distance from the cluster centers, and then choose the farthest distance from the smaller set.

Algorithm 2 Optimized Min-Max K-Means

input: Sample set $D=\{x_1, x_2, ..., x_m\}$, cluster number K

output: Initial cluster center $C=\{C_0, C_1, ..., C_{k-1}\}$

```
start
maxDis = 0
for i = 0; i < m; i ++
  for j = 0; j < m; j++
    Dis(i,j) = //x_i − x_j//
    if Dis(i, j) > max Dis
      max Dis = Dis(i, j), C_0 = x_i, C_1 = x_j
    endif
  endfor
endfor
for j = 1; j < k-1; j++
  for i = 1; i < m; i++
    d_{i,0} = //x_i − C_0//, d_{i,1} = //x_i − C_1//, ..., d_{i,j} = //x_j − C_j//
  endfor
  C_{j+1} = max[min(d_{i,0}, d_{i,1}, ..., d_{i,j-1})], i = 0,1,..., N−1
endfor
end
```

4 Adaptive differential evolution algorithm-based enhancing local search capacity (ESA_DE)

In this paper, the traditional differential evolution algorithm was improved, and an adaptive differential evolution algorithm (ESA_DE) was proposed to enhance the local search ability. It was applied to virtual machine migration to optimize the migration process in terms of network traffic, migration cost, performance interference, and energy consumption [9].

In order to achieve multi-objective optimization, four fitness functions were designed in this paper:

(1). The network flow generated in the process of virtual machine migration was related to the memory size and network routing of the virtual machine migration; this paper used $\text{Network}_{\text{fitness}} = \text{Min}(\text{NetworkFlow}_{V_i})$ as the fitness function of network traffic, NetworkFlow_{V_i} represented network traffic of virtual machine i; it could be expressed by formula (4).

$$\text{Network} = \sum_{i=1}^{m} dt_i \times dr_i \tag{4}$$

In formula (4), dt_i represented the data transmission size of the virtual machine i when it migrated, that is, the memory size of the virtual machine i. dr_i represented the length of network links when virtual machine i migrated, and the value is related to the network topology of cloud data center, the location of the source server, and the target server in the network topology. It could be calculated by formula (5):

$$dr_i = \left\{ \begin{array}{c} 3 \times 2, \text{routing through core switches} \\ 2 \times 2, \text{routing through aggregate switches} \\ 1 \times 2, \text{routing through edge switch} \end{array} \right\} \tag{5}$$

(2). The memory size and network bandwidth directly affected the migration cost; this paper used $\text{Cost}_{\text{fitness}} = \text{Min}(\text{MC})$ as the fitness function of the migration cost and MC represented the migration cost of all virtual machines.

$$\text{MC} = \sum_1^m \text{MC}_{V_i} = \sum_1^m 0.1 \times \sum_{t_0}^{t_0+t_{V_i}} U_{V_i} \tag{6}$$

In formula (6), MC_{V_i} represented the migration cost of virtual machine i, t_0 represented the migration start time of virtual machine i, t_{V_i} represented the total migration time of virtual machine i, and U_{V_i} represented the CPU utilization of servers occupied by migration of virtual machine i. It could be calculated by formula (7):

$$t_{V_i} = \frac{M_{V_i}}{B_{V_i}} \tag{7}$$

In formula (7), t_{V_i} represented the migration time of virtual machine i, M_{V_i} represented the memory size of virtual machine i, and B_{V_i} represented the network bandwidth occupied by virtual machine i.

(3). When the migration of virtual machines was completed, it would cause some performance interference to other virtual machines on this server, this paper used $\text{C}_{\text{fitness}} = \text{Min}(C_{V_i}^{PD})$ as the fitness function of the performance interference. It could be calculated by formula (8):

$$\text{C}_i^{PD} = 1 - a^{\frac{T_i^{PD} - T_i}{T_i}} \tag{8}$$

In formula (8), T_i represented the running time of virtual machine i on a single server, T_i^{PD} represented the running time of the virtual machine i and the virtual machine set on a single server, and a represented regulation parameters of performance interference.

(4). In the process of running the server, it would use $\text{E}_{\text{fitness}} = \text{Min}(E_i)$ as the fitness function of energy consumption. E_i represented the energy consumption of server i. It could be calculated by formula (9):

$$E_i = \sum_{t_1}^{t_n} P\left(u_i(t_j)\right) \tag{9}$$

In formula (9), $u_i(t_j)$ represented the CPU utilization of server i at time t_j, $P(u_i(t_j))$ represented the power of

server i at time t_j, and $P(u_i(t_j))$ could be expressed by formula (10).

$$P_i(u) = r_i \times P_i^{\max} + (1-r_i) \times P_i^{\max} \times u_i \tag{10}$$

Therefore, the multi-objective energy-saving optimization factor was shown in formula (11):

$$F = k_1 \mathrm{E}_{\mathrm{fitness}} + k_2 \mathrm{Network}_{\mathrm{fitness}} + k_3 \mathrm{Cost}_{\mathrm{fitness}} + k_4 \mathrm{C}_{\mathrm{fitness}} \tag{11}$$

In formula (11), k_1, k_2, k_3, k_4 was the balance factor of each factor, its scope belonged to [0,1], $k_1 + k_2 + k_3 + k_4 = 1$, and the parameters were used to adjust and control the influence of various factors on the comprehensive fitness.

In the process of research, we set the size of population to N, the number of virtual machines migration to D, the number of servers to M, and randomly generated N individuals whose length was D. Individuals represented a virtual machine migration scheme, in which the placement value of each virtual machine was taken from $[1, M]$. The maximum iterations number was K, the mutation ratio factor $\phi \in [0, 2]$, and the crossover probability factor $P_c \in [0, 1]$. The first generation of the ith individual was represented by:

$$x_k(0) = \left(x_{1j}^k(0), x_{2j}^k(0), \ldots, x_{ij}^k(0), \ldots, x_{dj}^k(0)\right) \tag{12}$$

In formula (12), $x_k(0)$, $(k = 1, 2, \ldots, N)$ represented the kth virtual machine of the 0th generation that needed to be migrated and $x_{ij}^k(0), (i = 1, 2, \ldots, D; j = 1, 2, \ldots, M)$ represented the migration of the ith virtual machine to the jth server; its range of values was randomly generated by formula (13).

$$x = x_{j\_\min} + rand(0,1) \times \left(x_{j\_\max} - x_{j\_\min}\right) \tag{13}$$

In formula (13), $x_{j\_\min}$ represented the minimum value of vector and $x_{j\_\max}$ represented the maximum value of vector, mapping to $[1, M]$.

According to the differential evolution algorithm, different virtual machine migration schemes $x_{t_1}(g)$, $x_{t_2}(g)$, and $x_{t_3}(g)$ were randomly selected from the g generation population. The mutation operation was carried out to generate new individuals of the population increasing the diversity of the population. The variation scaling factor of the difference process could be expressed as formula (14):

$$\mathrm{vector} = \phi(x_{t_1}(g) - x_{t_2}(g)) \tag{14}$$

In formula (14), $x_{t_1}(g) - x_{t_2}(g)$ represented the differential vector and vector represented vectors by weighted; constituting the variation vector by the third individual $x_{t_3}(g)$, a new virtual machine migration scheme was generated, and its expression was shown in formula (15):

$$v_t(g+1) = x_{t_3} + \mathrm{vector} \tag{15}$$

In formula (15), $v_t(g+1)$ represented newly generated individuals, $t = 1, 2, 3, \ldots, N$.

In order to increase the diversity of the population, cross-over operation was introduced. New mutated individuals $v_t(g+1)$ and original individuals $x_t(g)$ were crossed to produce new individuals $u_t(g+1) = (u_{1j}^t(g+1), u_{2j}^t(g+1), \ldots, u_{dj}^t(g+1))$. The expression of the crossover process was shown in formula (16):

$$u_{ij}^t(g+1) = \begin{cases} v_{ij}^t(g+1), & rand(i) \le P_c ori = rand(t) \\ x_{ij}^t(g), & rand(i) > P_c ori \neq rand(t) \end{cases} \tag{16}$$

In formula (16), $j = (1, 2, \ldots, M)$, $rand\,(i)$ represented as a random number between (0, 1), cross probability factor P_c, and $rand(t) \in [1, N]$.

Then, the population entering the next cycle was selected through the selection operation, and the fitness function of the cross-generated individual $u_t(g+1)$ was compared with that of the target individual $x_t(g)$. The expression of the fitness function was as follows:

$$x_t(g+1) = \begin{cases} u_t(g+1), F(u_t(g+1)) > F(x_t(g)) \\ x_t(g), \mathrm{others} \end{cases} \tag{17}$$

In formula (17), $F(x)$ represented the fitness function which individual needed to satisfy and $x_t(g+1)$ represented individuals of the next generation.

By comparing the fitness values, the better one would choose to enter the next generation of iteration process and ultimately get the optimal solution [10]. The specific process is shown in Algorithm 3:

Algorithm 3 ESA_DE

input: Population size: N, Number of servers: M, Number of virtual machines: D
The maximum number of iterations: K, Variation factor: $\varphi \in [0,2]$
Cross probability factor: $P_c \in [0,1]$

output: Optimal scheme set for virtual machine migration

1. start
2. Number of iterations: g= 0, positive integer: i = 1,j=1;
3. for i = 0; i< M; i++
4. for j = 0; j < D; j++
5. $x = x_{j\_\min} + rand(0,1) * (x_{j\_\max} - x_{j\_\min})$
6. endfor
7. endfor
8. while($g \le k$)
9. Random selection of three individuals $x_{t_1}(g), x_{t_2}(g), x_{t_3}(g)$ as weight φ, mutation generates new individuals.
10. Comparing random number with cross probability factor, crossed $v_t(g+1)$ and $x_t(g)$
11. The new individuals $u_t(g+1) = (u_{1j}^t(g+1), u_{2j}^t(g+1), \ldots, u_{dj}^t(g+1))$
12. if $F(u_t(g+1)) > F(x_t(g))$
13. $x_t(g+1) = u_t(g+1)$
14. else
15. $x_t(g+1) = x_t(g)$
16. endif
17. g= g + 1;
18. Local search is performed according to algorithm 4
19. endwhile
20. return Optimal scheme set for virtual machine migration
21. end

In the experiment, it was found that the convergence speed of the traditional differential evolution algorithm was slower and the search ability of the algorithm was lower. In the process of cross-mutation, infeasible individuals or overload were prone to occur. In order to avoid this situation, this paper improved the traditional differential evolution algorithm by adding a local search operator. The specific process is shown in Algorithm 4:

Algorithm 4 Enhanced-filter operator

input: Local search step size: δ_i, number of local iterations: T, population size: N

Individual population: $x_i(g)$, step size reduction factor: ε

output: The better individual of next generation

```
start
The population individual: i = 0, the iteration counter: t= 0, g-th generation;
while( i ≤ N )
  while( t ≤ T )
    Random variable Δx, Δx ∈ [−δ_t, δ_t]
    x_new(g) = x_i(g) + Δx
    // H():whether the new population is feasible
    if H(x_new(g)) = true and H(x_i(g)) = false)
      x_i(g) = x_new(g)
    endif
    if ( H(x_new(g)) = true and H(x_i(g)) = true)
      if( F(x_i(g)) < F(x_new(g)) )
        x_i(g) = x_new(g)
      endif
    endif
    δ_t = δ_t * ε
    t=t+1
  endwhile
  i = i + 1
endwhile
end
```

In the above process, a local search was performed on the current optimal solution. Filter results in an infeasible or overloaded migration program so that the solution of its neighborhood met the condition of migration and was closer to the optimal solution.

5 Simulation experiment and result analysis

CloudSim4.0, a cloud computing simulation software, was used to simulate the proposed model and algorithm. The proposed server load detection algorithm (SVR_GA), virtual machine classification selection algorithm (K-Means-OMM), and server selection algorithms based on load balancing target (ESA_DE) were compared with classical algorithms.

5.1 Experimental environment and parameter configuration

5.1.1 Experimental environment

1. Experimental server configuration:
   (1) CPU: Intel® Core™ i7 4770 3.4 GHz
   (2) Memory(RAM): 8.0G
   (3) Hard disk: 1024G
   (4) Operating System: Windows 7 (64 bits)
2. Development environment: Eclipse 4.5.1,JDK1.8.0_111
3. CloudSim version: CloudSim4.0

5.1.2 Server and virtual machine parameter

1. Server configuration
   In this paper, seven types of servers are involved in the experiment. The MIPS of server ranges from 1.86GHZ to 3.067GHz, the core number is 2-12, the memory size is 4G-16G, the bandwidth is 1000Mbit/s, and the hard disk size is 50G-320G.
2. Virtual machine configuration
   In this paper, four types of virtual machines are involved in the experiment. The MIPS of server ranges from 0.5GHZ to 2.5GHz, the core number is 1, the memory size is 613M-1740M, the bandwidth is 100Mbit/s, and the hard disk size is 2.5G.

5.2 Experimental result

5.2.1 Load forecasting experiment

Figures 2, 3, and 4 are comparative experiments of CPU utilization between the server load detection algorithm (SVR_GA) and Bayesian Ridge Regression [11], Decision Tree Regression [12], and Support Vector Regression (SVR RBF Model) in the process of virtual machine scheduling.

Figure 2 is the CPU utilization which predicts data comparison of the four algorithms with the actual CPU utilization data when the server ID was 3. From the graph, it could be seen that the predicted value of the support vector regression Gaussian model deviated from the actual value greatly, and the decision tree model and the Bayesian ridge regression model deviated little from the actual value, while the support vector regression Gaussian model optimized by the genetic algorithm deviated least from the actual value.

Figure 3 shows the error comparison between the predicted CPU utilization and the actual CPU utilization of the four algorithms when the server ID was 3. The prediction error range of CPU utilization based on Bayesian ridge regression model was − 0.42~0.23. The prediction error range of CPU utilization based on the decision tree model was − 0.35~0.32. The prediction error range of CPU utilization of the support vector regression Gauss model was − 0.53~0.25. The prediction error range of CPU utilization of the support vector regression Gauss model optimized by the genetic algorithm was − 0.22~0.24. Among the four algorithms, the CPU utilization prediction error range of the proposed algorithm was the smallest.

Figure 4 is a boxplot of the error between the predicted CPU utilization and the actual CPU utilization of the four algorithms when the server ID was 3. From the

graph, it could be clearly seen that the minimum error of the Bayesian ridge regression model and decision tree model is close to 0, the maximum value is near 0.2, and there were a few outliers, which showed that the prediction error of the two algorithms was small and the stability is high; the minimum error of support vector regression Gaussian model was close to 0, the maximum value is less than 0.1, but there were many outliers, which showed that the algorithm had small error and high stability. Although CPU utilization could be predicted accurately, it lacked stability due to the large number of outliers. The maximum and minimum errors of the support vector regression Gauss model optimized by the genetic algorithm were close to 0, and there were fewer outliers. It showed that the optimization algorithm could improve the accuracy of the support vector regression Gauss model and increase the stability of the algorithm.

5.2.2 Virtual machine classification experiment

In this paper, three initial clustering center selection schemes were compared. The first scheme used the random algorithm to select the initial clustering center; the second scheme used the min-max algorithm to select the initial clustering center; and the third scheme was optimized on the basis of the min-max algorithm. In the above, the performance interference of virtual machines could be divided into two categories: high-performance interference and low-performance interference. The server load state could be divided into three categories: high load, low load and normal load. Therefore, the virtual machines were divided into six categories in this experiment. The experimental results are shown in Fig. 5.

Figure 5 classifies 280 sample points (virtual machines). The X coordinate represented the remaining CPU utilization of the server after the virtual machine migration, and the Y coordinate represented the remaining performance interference after the virtual machine migration. The sample points in the graph were normalized and could not represent the actual values.

From Fig. 5a, we could see that the random selection algorithm chose the cluster center to divide the purple sample points into the first category, the upper and lower parts. The upper and lower parts were far away. The sample points in the red ellipse were close to the sample points in the fifth category. The random selection algorithm had some problems in classifying the purple sample points.

From Fig. 5b, we could see that the min-max algorithm chose the clustering center to divide the sample points in the red ellipse into two categories, and the distance between these sample points was relatively close, which should be divided into one category. The min-max algorithm had some problems in dividing these points.

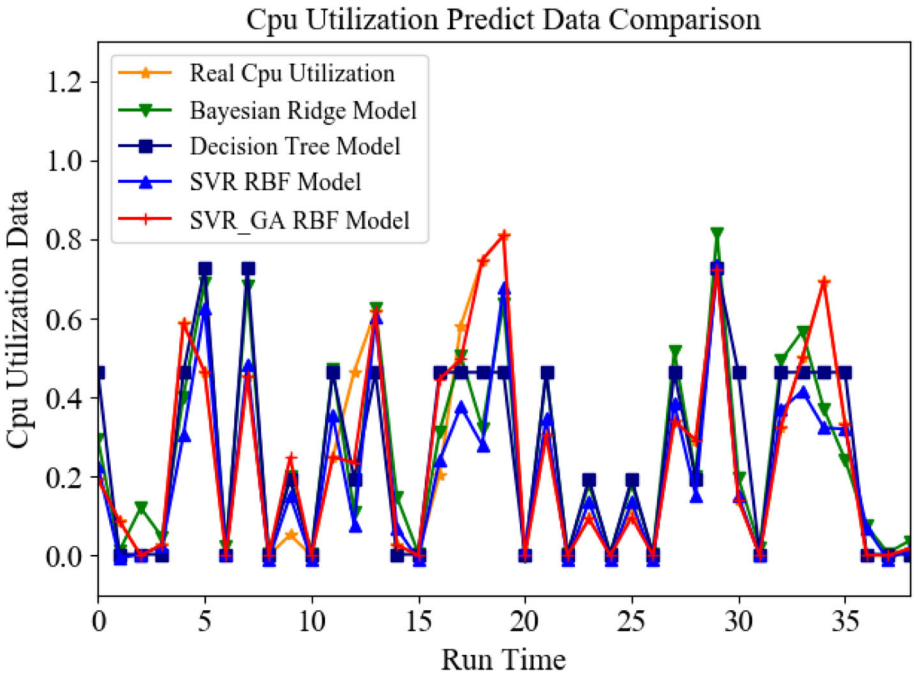

Fig. 2 Server load prediction comparison

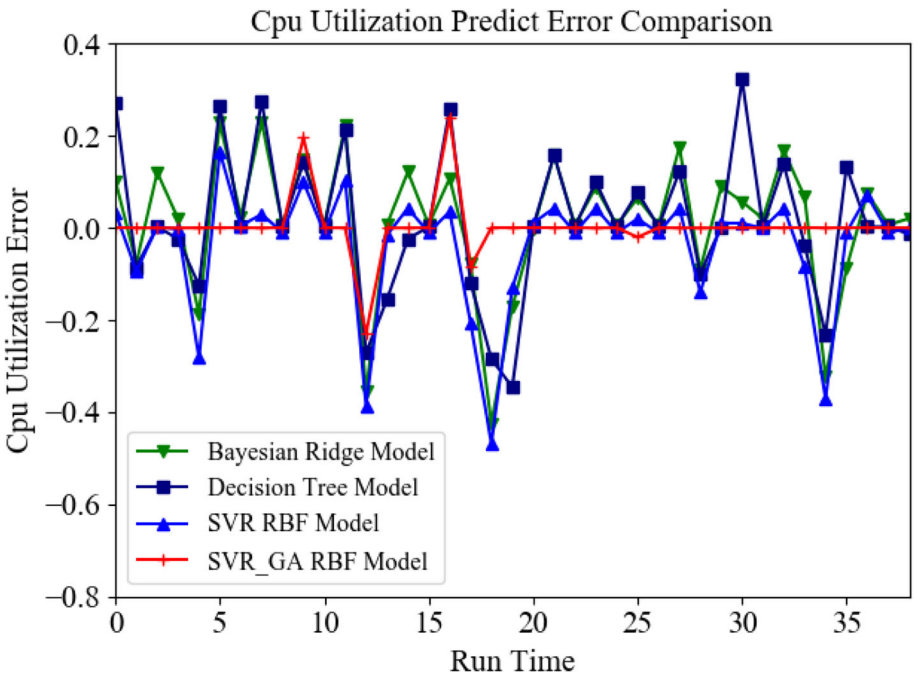

Fig. 3 Comparison of server load prediction errors

From Fig. 5c, the improved min-max algorithm proposed in this paper was more suitable for the division of initial clustering centers. Therefore, this paper took the optimized min-max algorithm as the initial clustering center selection scheme of the *k*-means clustering algorithm.

5.2.3 Load balancing experiment

5.2.3.1 Migration cost Figure 6 is an experimental comparison of the total cost of virtual machine migration between ESA_DE and IQR_MC, LRR_MMT and MAD_RS under different number of tasks. From

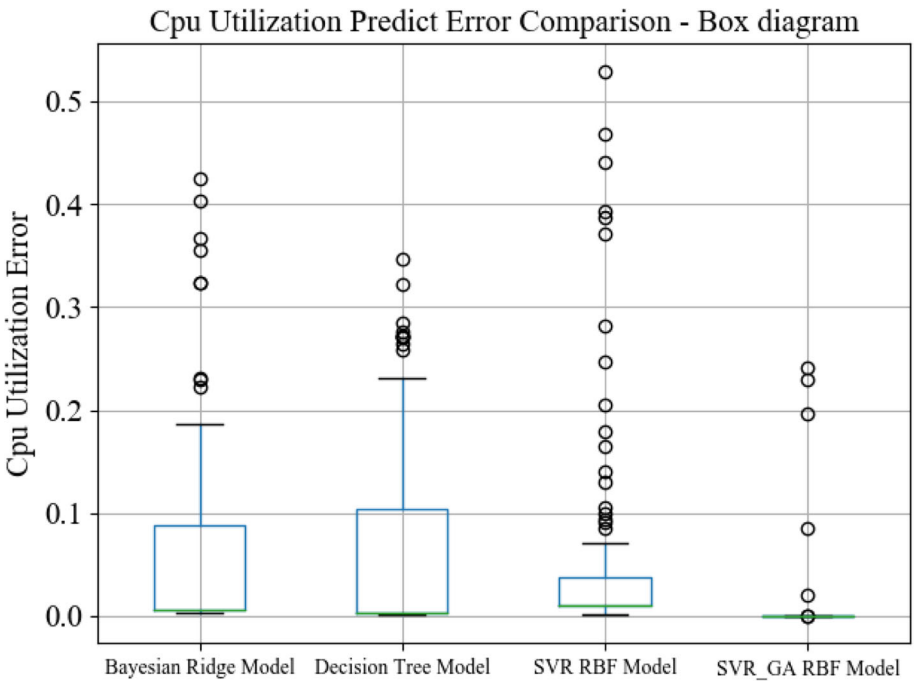

Fig. 4 Comparison of server load prediction errors-box diagram

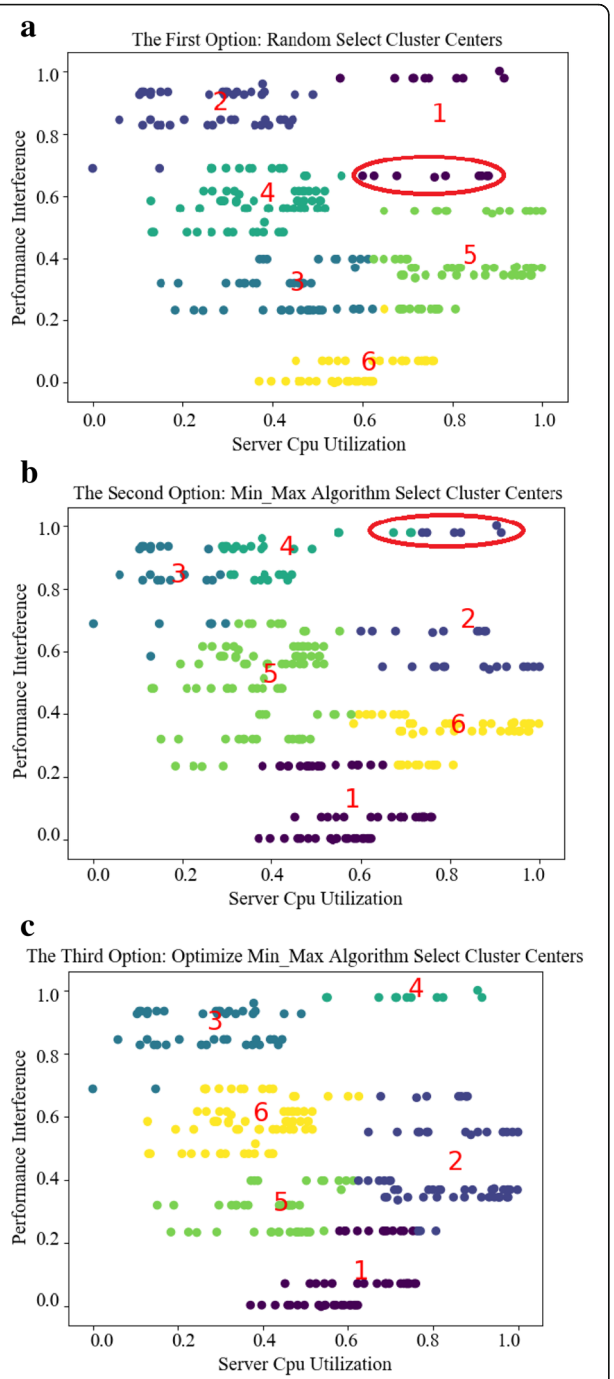

Fig. 5 **a** Random algorithm for selecting initial clustering centers. **b** Min-max algorithm for selecting initial clustering centers. **c** Optimizing min-max algorithm for selecting initial clustering centers

the graph, it could be seen that the ESA_DE algorithm proposed in this paper could greatly reduce the migration cost of virtual machines under each task number. The experimental data showed that the ESA_DE algorithm was 80.56~ 87.22% lower than the IQR_MC algorithm, 64.24~ 80.01% lower than the LRR_MMT algorithm, and 79.44~ 85.95% lower than MAD_RS algorithm.

Figure 7 is an experimental comparative analysis of the migration cost of the ESA_DE algorithm and IQR_MC, LRR_MMT, and MAD_RS at different time. From the graph, we could see that in the initial stage of task execution, the migration cost of the four algorithms were all in a high position, which showed that the load of the data center was unbalanced at this time, and the load needed to be balanced by virtual machine migration. Over time, the migration cost of the four algorithms was gradually reduced, indicating that the load of the data center was in a relatively balanced state. The virtual machine migration cost of the ESA_DE algorithm is the lowest among the four algorithms.

5.2.3.2 Performance interference Figure 8 is an experimental comparison of the ESA_DE algorithm and IQR_MC, LRR_MMT, and MAD_RS algorithms for virtual machine performance interference under different number of tasks. It was obvious from the graph that the ESA_DE algorithm proposed in this paper could minimize the performance interference between virtual machines. The experimental data showed that the ESA_DE algorithm was 81.91~ 88.36% lower than the IQR_MC algorithm, 78.45~ 86.24% lower than the LRR_MMT algorithm, and 80.02~ 87.05% lower than the MAD_RS algorithm.

Figure 9 is an experimental comparison and analysis of the performance interference between ESA_DE and IQR_MC, LRR_MMT, and MAD_RS at different time. It could be clearly seen from the graph that in the initial stage, the performance interference of the four algorithms was very large, which was caused by more virtual machine migration in the initial stage; with the gradual reduction of the migration number, the performance interference between virtual machines also decreased; and the performance interference of the ESA_DE algorithm proposed in this paper was the smallest among the four algorithms.

5.2.3.3 Network flow Figure 10 is an experimental comparative analysis of the total network flow between the ESA_DE algorithm and IQR_MC, LRR_MMT, and MAD_RS algorithms under different number of tasks. Compared with other three algorithms, the ESA_DE algorithm proposed in this paper effectively reduced the total network flow of data center. From the experimental data, the total network flow of the ESA_DE algorithm was 6.1~ 7.87% of the IQR_MC algorithm, 12.52~ 16.69% of the LRR_MMT algorithm, and 6.43~ 8.56% of the MAD_RS algorithm.

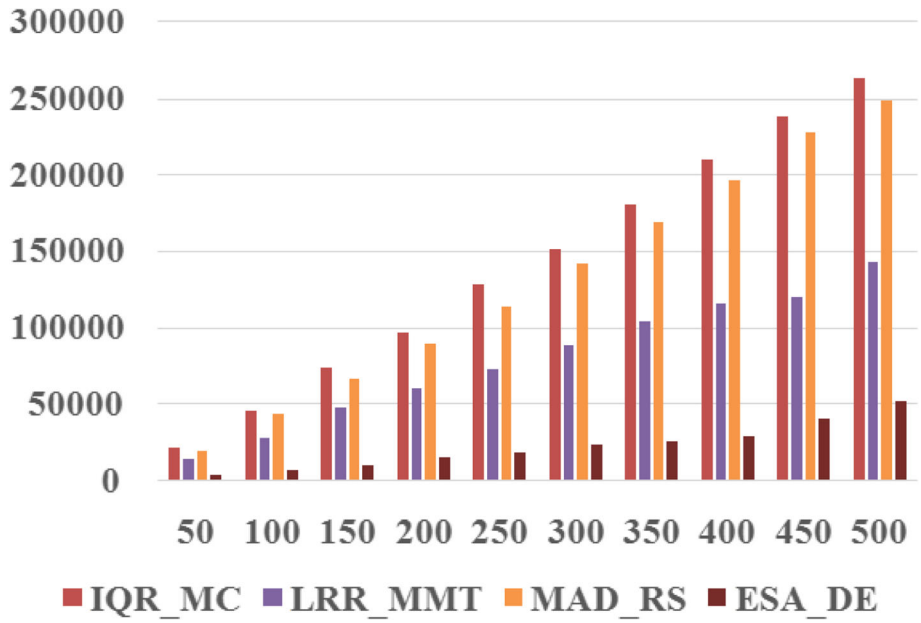

Fig. 6 Total migration cost of the virtual machine under different task numbers

Figure 11 is a comparative experiment of the average network flow between ESA_DE and IQR_MC, LRR_MMT, and MAD_RS at different time. As could be seen from the graph, the average network flow of the four algorithms was IQR_MC, MAD_RS, LRR_MMT, and ESA_DE in turn. In the vicinity of 5000, 10,000, 20,000, and other time points, the average network flow showed a sudden increase, which was caused by the migration of virtual machines due to server overload and the completion of virtual machine tasks. In the whole operation cycle, no matter how the virtual machine migrated, the average network flow of the ESA_DE algorithm was the lowest.

5.2.3.4 CPU utilization Figure 12 is a comparative experiment of the average CPU utilization of the ESA_DE algorithm and IQR_MC, LRR_MMT and MAD_RS under different task numbers. From the experimental data, it could be concluded that the average CPU utilization of the IQR_MC algorithm ranged from 39.06 to 41.36%, that of the LRR_MMT

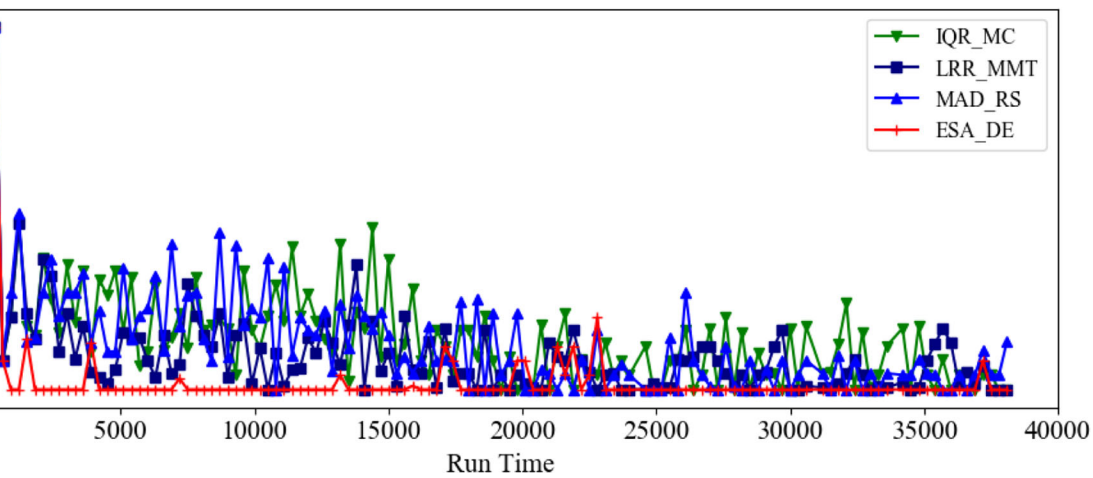

Fig. 7 Average migration cost of the virtual machine at different time

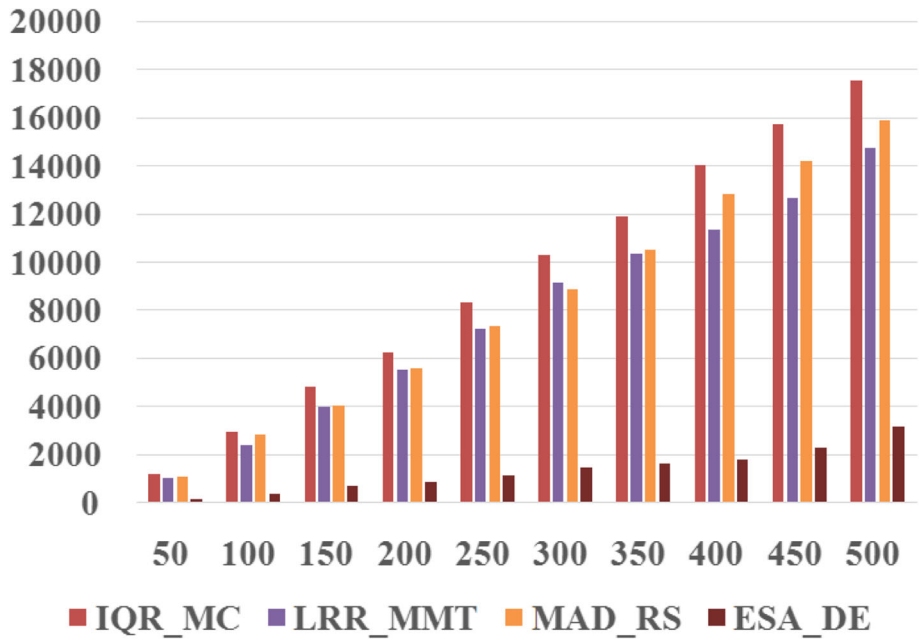

Fig. 8 Total performance interference of virtual machine under different task numbers

algorithm ranged from 44.17 to 46.76%, and that of the MAD_RS algorithm ranged from 43.52 to 46.13%. The average CPU utilization of the ESA_DE algorithm proposed in this paper ranged from 73.19 to 84.87%. With the increase of the number of tasks, the average CPU utilization of the four algorithms is also increasing. This paper proposes that the average CPU utilization of the ESA_DE algorithm is 87.38~ 106.8% higher than that of the random algorithm and 65.69~ 83.22% higher than that of the LRR_MMT algorithm. It is 68.19~ 86.68% higher than the MAD_RS algorithm. The average CPU utilization of the ESA_DE algorithm proposed in this paper is basically maintained above 80%. This is because the ESA_DE algorithm balances the load status of each server in the cloud data center through the virtual machine migration strategy.

5.2.3.5 Virtual machine migration number Figure 13 is an experimental comparison of the proposed ESA_DE algorithm on virtual machine migration number with IQR_MC, LRR_MMT, and MAD_RS under different task numbers. It can be seen from the figure that compared with the other three algorithms, the virtual machine migration number of the ESA_DE algorithm proposed in this paper is effectively reduced. Experimental data shows that the number of virtual machine migrations of the ESA_DE algorithm is 93.05~ 94.5% lower than that of the IQR_MC algorithm, 88.79~ 91.56% lower than that of the

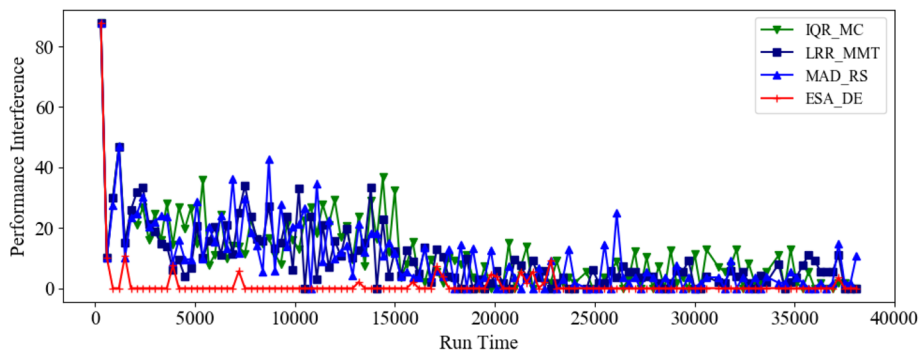

Fig. 9 Performance interference of virtual machine migration at different time

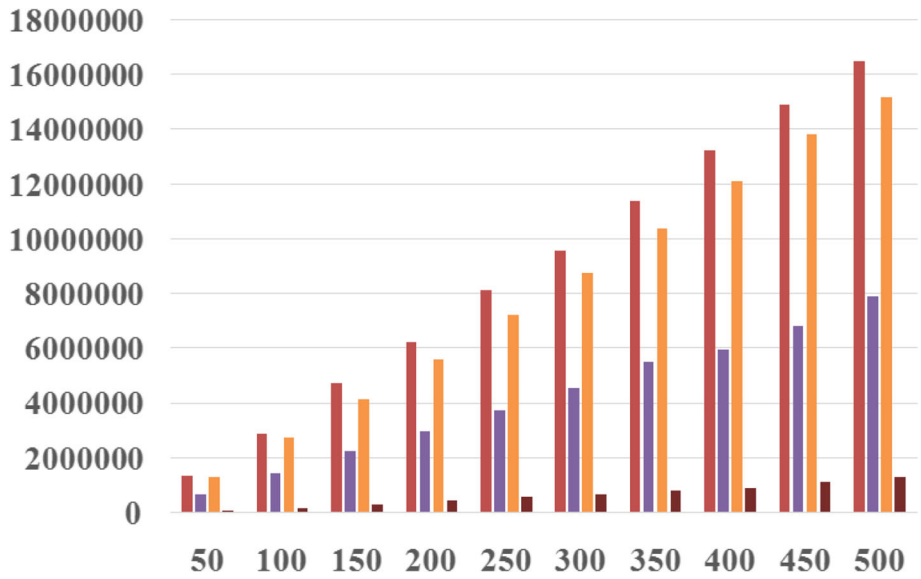

Fig. 10 Total network flow under different task numbers

LRR_MMT algorithm, and 92.22~ 94.17% lower than that of the MAD_RS algorithm.

5.2.3.6 Data center energy consumption Figure 14 is a comparison of the data center energy consumption experiments of the ESA_DE algorithm and the IQR_MC, LRR_MMT, and MAD_RS algorithms under different task numbers. It can be seen from the experimental results that the data center energy consumption of the four algorithms is IQR_MC, MAD_RS, LRR_MMT, and ESA_DE algorithms from large to small. In this paper, the data center energy consumption of the ESA_DE algorithm is reduced by 42.44~ 49.13% compared with that of the IQR_MC algorithm, 36.8~ 42.54% lower than that of the LRR_MMT algorithm, and 37.29~44.36% lower than that of the MAD_RS algorithm.

6 Conclusion

From the perspective of green scheduling, this paper proposed the load forecasting algorithm (SVR_GA) based on machine learning algorithm, virtual machine classification algorithm (K-Means-OMM), and multi-objective optimization algorithm (ESA_DE) based on heuristic. The SVR_GA algorithm could accurately predict the CPU utilization of servers, and the prediction error was only − 0.22~0.24; the K-means-OMM algorithm could accurately classify virtual machines, which was better than random selection and the min-max algorithm on initialing clustering centers; compared with the IQR_MC, LRR_MMT, and MAD_RS algorithms, the migration cost, network flow, and performance interference of the ESA_DE algorithm proposed in this paper were lower than others. The average utilization of data center is higher than that of other algorithms, reaching about

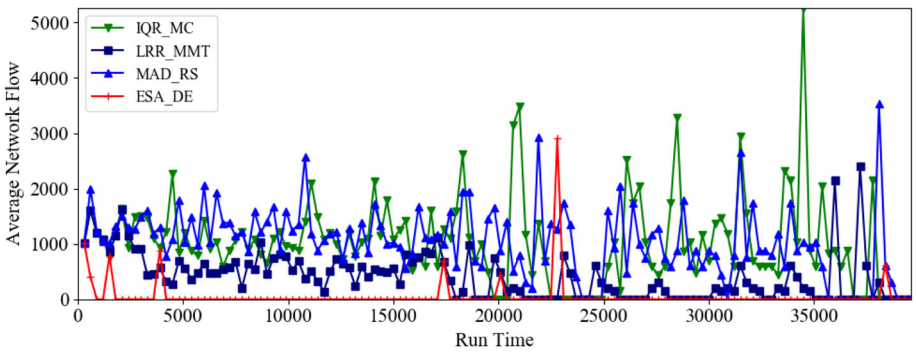

Fig. 11 Average network flow at different time

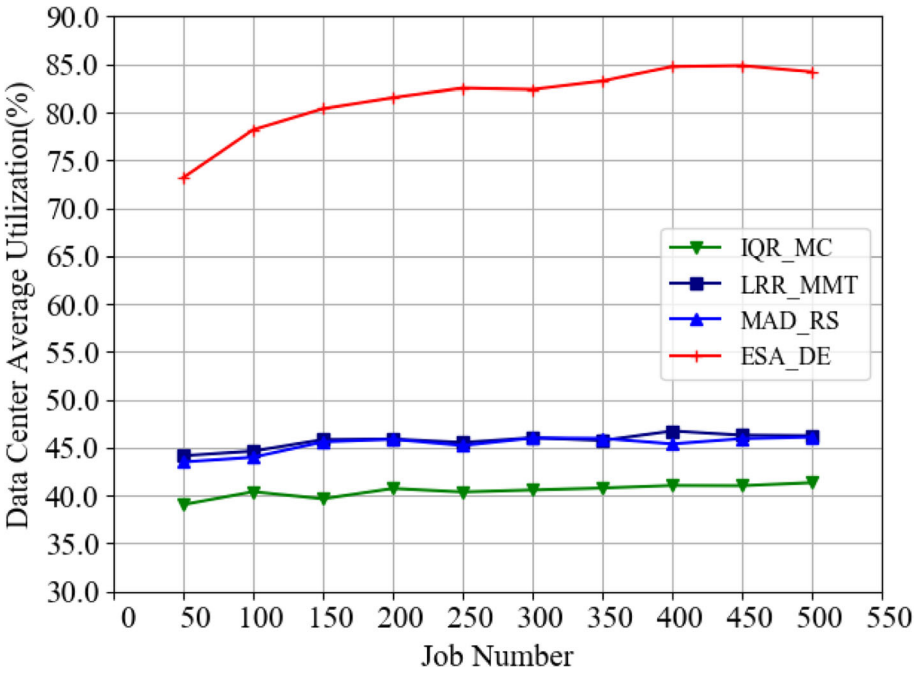

Fig. 12 Average CPU utilization of the data center under different task numbers

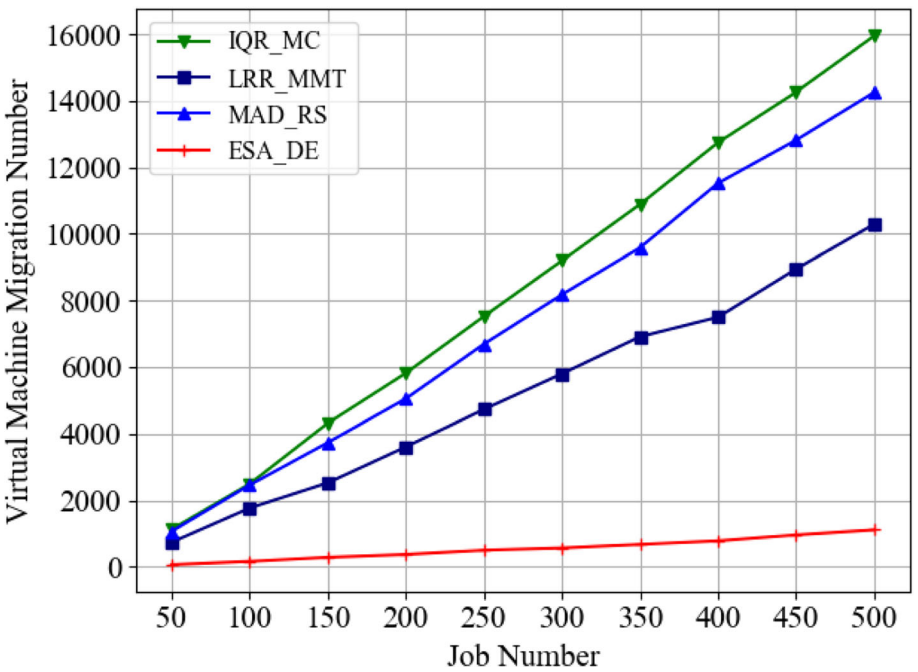

Fig. 13 Virtual machine migration number under different task numbers

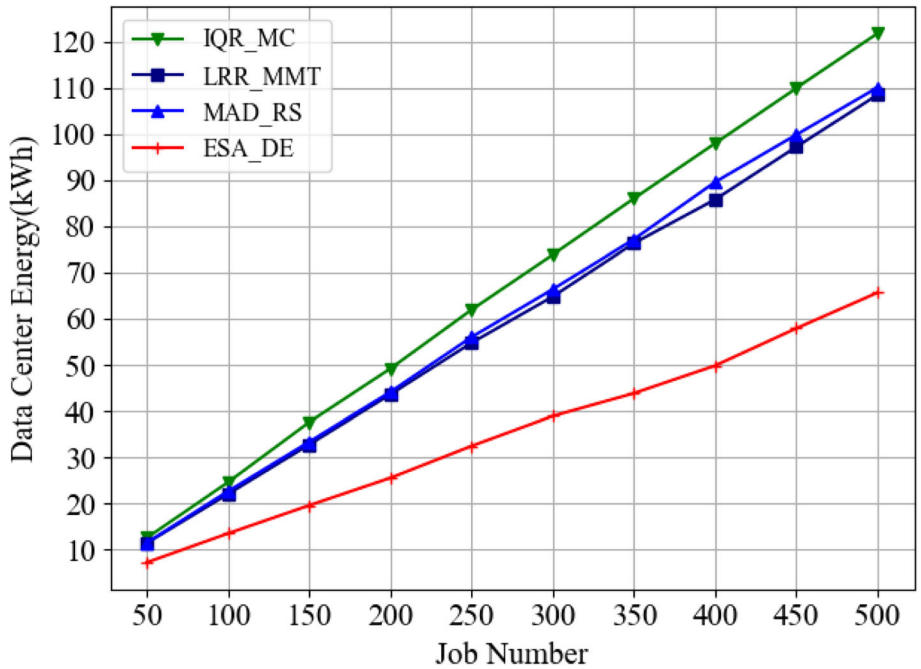

Fig. 14 Data center energy consumption under different task numbers

80%; the maximum number of virtual machine migration is reduced by 94.5%; the maximum energy consumption of data center is reduced by 49.13%; and the goal of load balancing is achieved.

Abbreviations

ESA_DE: Adaptive differential evolution algorithm-based enhancing local search capacity; IQR_MC: Inter quartile range and maximum correlation algorithm; K-Means-OMM: *k*-means clustering algorithm-based optimized min-max; LRR_MMT: Local regression robust and minimum migration time algorithm; MAD_RS: Median absolute deviation and random selection algorithm; MO-LB: Multi-objective load balancing; QoS: Quality of service; SDN: Software-defined networks technology; SLA: Service-level agreement; SVR: Support vector regression; SVR_GA: Support vector regression load forecasting algorithm based on genetic algorithm optimization; VMCUP: Virtual machine consolidation algorithm with usage prediction

Acknowledgements

This work was supported in part by the Science and Technology project of "13th Five-Year" planning of the Education Department of Jilin Province (JJKH20180750KJ and JJKH20190593KJ), the Key Science and Technology Project of Jilin Province (20160204019GX), and the CERNET Innovation Project (NGII20180127).

Authors' contributions

XS and DL designed the algorithm. XS wrote this paper. XS and HW did the experimental tests. LL optimized the algorithm. LL and H-wY checked the whole paper and figures. All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Publisher's Note

Author details

[1]College of Computer Science and Technology, Changchun University of Science and Technology, Changchun 130022, China. [2]Jilin Provincial Institute of Education, Changchun 130022, China.

Received: 22 February 2019 Accepted: 25 April 2019
Published online: 17 June 2019

References

1. L. Yu, L. Chen, Z. Cai, et al., Stochastic load balancing for virtual resource management in datacenters. IEEE Trans. Cloud Comput. 1–14 (2016)
2. M. Elrotub, A. Gherbi, Virtual machine classification-based approach to enhanced workload balancing for cloud computing applications. Procedia Comput. Sci. **130**, 683–688 (2018)
3. F. Ramezani, J. Lu, J. Taheri, et al., A multi-objective load balancing system for cloud environments. Comput. J. **60**(9), 1316–1337 (2017):comjnl; bxw109v2
4. N.T. Hieu, M., D. Francesco, A. Ylajaaski, in IEEE International Conference on Cloud Computing. Virtual machine consolidation with usage prediction for energy-efficient cloud data centers. IEEE, 750-757 (2015)
5. W. Chen, Z. Shang, X. Tian, et al., Dynamic server cluster load balancing in virtualization environment with OpenFlow. Int. J. Distributed Sensor Networks **11**(7), 531538 (2015)
6. V. Tyagi, T. Kumar, ORT broker policy: reduce cost and response time using throttled load balancing algorithm ☆. Procedia Comput. Sci. **48**, 217–221 (2015)
7. G. Cao, L. Wu, Support vector regression with fruit fly optimization algorithm for seasonal electricity consumption forecasting. Energy **115**, 734–745 (2016)
8. X. Wang, Y. Bai, The global Minmax k-means algorithm. SpringerPlus **5**(1), 1665 (2016)
9. Z.H. Zhan, X.F. Liu, H. Zhang, et al., Cloudde: a heterogeneous differential evolution algorithm and its distributed cloud version. IEEE Trans. Parallel Distrib. Syst. **28**(3), 704–716 (2017)
10. A. Leivadeas, C. Papagianni, S. Papavassiliou, Efficient resource mapping framework over networked clouds via iterated local search-

based request partitioning. IEEE Trans. Parallel Distrib. Syst. **24**(6), 1077–1086 (2013)
11. W. Gao, Q. Chen, Y. Ge, et al., The probabilistic model and forecasting of power load based on variational Bayesian expectation maximization and relevance vector machine. Wirel. Pers. Commun. **102**(4), 3041–3053 (2018)
12. H. Wang, T. Wang, Y. Zhou, et al., Information classification algorithm based on decision tree optimization. Clust. Comput. (2018)