Tsinghua Science and Technology

Volume 26 | Issue 1 Article 6

2021

A Multi-Objective Optimization Method of Initial Virtual Machine Fault-Tolerant Placement for Star Topological Data Centers of Cloud Systems

Wei Zhang
the School of Software Engineering, Tongji University, Shanghai 201804, China.

Xiao Chen
the School of Software Engineering, Tongji University, Shanghai 201804, China.

Jianhui Jiang
the School of Software Engineering, Tongji University, Shanghai 201804, China.

TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 09/10 pp95–111
DOI: 10.26599/TST.2019.9010044
Volume 26, Number 1, February 2021

A Multi-Objective Optimization Method of Initial Virtual Machine Fault-Tolerant Placement for Star Topological Data Centers of Cloud Systems

Wei Zhang, Xiao Chen, and Jianhui Jiang*

Abstract: Virtualization is the most important technology in the unified resource layer of cloud computing systems. Static placement and dynamic management are two types of Virtual Machine (VM) management methods. VM dynamic management is based on the structure of the initial VM placement, and this initial structure will affect the efficiency of VM dynamic management. When a VM fails, cloud applications deployed on the faulty VM will crash if fault tolerance is not considered. In this study, a model of initial VM fault-tolerant placement for star topological data centers of cloud systems is built on the basis of multiple factors, including the service-level agreement violation rate, resource remaining rate, power consumption rate, failure rate, and fault tolerance cost. Then, a heuristic ant colony algorithm is proposed to solve the model. The service-providing VMs are placed by the ant colony algorithms, and the redundant VMs are placed by the conventional heuristic algorithms. The experimental results obtained from the simulation, real cluster, and fault injection experiments show that the proposed method can achieve better VM fault-tolerant placement solution than that of the traditional first fit or best fit descending method.

Key words: cloud computing; virtual machine placement; fault tolerance; multi-objective optimization; heuristic ant colony algorithm

1 Introduction

The structure of a cloud computing system includes the organization, unified resource, platform, and application layers[1]. Virtualization is the most critical technology in the unified resource layer, which can improve the resource utilization ratio of physical servers and reduce the cost of building data centers. However, statistical analysis of Amazon web services shows that the top three factors of server downtime are power supply, storage, and Virtual Machines (VMs)[2]. Therefore, VM fault-tolerant technology and data center reliability management are becoming increasingly important.

Two kinds of VM fault-tolerant techniques are widely used. One is based on the checkpoint-based and log-based rollback methods, which means that the VMs placed on the failed node in a cloud computing system will migrate online to another node[3–8]. The other is based on the primary – backup model with the idea of incremental checkpoints. When the system starts running, the primary machine has the same data as the backup machine. The primary machine processes the compute tasks, generates the output while it is running, and synchronizes the modified data to the backup machine at a given time interval. When the primary machine breaks down, the backup machine takes over and processes the compute tasks[9, 10].

Static placement and dynamic management are two types of VM management methods in data centers. VM static placement focuses on the deployment of a large number of VMs on a given number of physical machines (nodes) at the beginning[11]. VM dynamic

• Wei Zhang, Xiao Chen, and Jianhui Jiang are with the School of Software Engineering, Tongji University, Shanghai 201804, China. E-mail: 1910134@tongji.edu.cn; 21_chenxiao_study@tongji.edu.cn; jhjiang@tongji.edu.cn.
* To whom correspondence should be addressed.
Manuscript received: 2019-06-27; accepted: 2019-08-28

management uses the VM online migration technology while the system is running. The migration target node is selected on the basis of the real-time load. The VMs are remapped to the physical nodes under the condition of ensuring the least migration cost[12–15].

The VM static placement problem is a matching problem between the resource requested by the VMs and the resource provided by the physical nodes. The resources include the Central Processing Unit (CPU), memory, disk, and network bandwidth. Mapping requires the consideration of certain factors, such as quality of user services, resource utilization ratio, and power consumption rate in data centers. VM static placement can be regarded as a packing problem, which is NP-hard[16].

Most of the existing static placement methods of VMs considered only a single constraint. For example, in Ref. [17], the quality of user service as well as the Service-Level Agreement (SLA) violation rate, were measured and the SLA violation rate while the VMs were being placed was minimized. The SLA is an agreement between a cloud user and a web service provider that sets out a range of parameters, such as service description, priority, and service level. Other methods include the improvement of the resource utilization ratio of physical nodes to reduce resource loss[18] and the combination of resource placement with power management by shutting down zero or low workload physical nodes[19–21]. Several methods were used to solve the multiple target optimization problem. For example, in Ref. [22], the resource utilization ratio, power consumption, and thermal power consumption were considered. In Ref. [23], a method to minimize the cost of mapping VMs to physical nodes under the influence of the SLA violation rate, resource utilization ratio, and power consumption was proposed. In Ref. [24], genetic algorithms were used to study the mapping strategy of VMs to physical nodes that can minimize resource usage and the number of physical nodes. However, the reliability problem was not considered in the previously mentioned works.

At present, several studies have been conducted to investigate the fault-tolerant placement of VMs. In Ref. [25], a method to determine the minimum number of physical nodes when multiple services were placed on different VMs were proposed. The method ensured that the system can tolerate multiple physical node failures, but it only considered the influence of response time. In Ref. [26], the influence of VM placement on the reliability of applications was analyzed, and the results indicated that the VMs providing different services should be placed on different physical nodes on the basis of their characteristics. However, other factors, such as resource utilization ratio and SLA violation rate, were not considered. In Ref. [27], a method to place VMs when multiple physical nodes failed was proposed, and it can ensure that a certain number of VMs can run normally at any time. However, the cost of VM placement was not considered. Other studies also considered how to ensure high availability of cloud computing systems when placing VMs[28–31].

For the reliability problem in cloud computing systems, this study focuses on how to map different types of VMs to physical nodes under the influence of five factors, i.e., SLA violation rate, resource remaining rate, power consumption rate, failure rate, and fault tolerance cost. A heuristic ant colony algorithm is proposed to solve the multi-objective optimization problem of initial VM fault-tolerant placement for star topological data centers of cloud systems.

The remainder of this paper is organized as follows: Section 2 introduces the system structure and problem description of VM fault-tolerant management. Section 3 describes a new multi-objective optimization algorithm of initial VM fault-tolerant placement. Section 4 presents the experimental results and analysis of the simulation, real cluster, and fault injection experiments. Section 5 concludes the paper.

2 VM Fault-Tolerant Placement Management System

2.1 System structure

In this study, we assume that the data centers of cloud computing systems are star topological; all of the physical nodes are connected by a switch; and the network connection is reliable.

The unified resource layer of a cloud computing system is a cluster of many physical nodes, which contains b racks, i.e., rack_1, rack_2, …, rack_b, and rack_i ($1 \leqslant i \leqslant b$) is placed on node_{i1}, node_{i2}, …, node_{in_i}. VMs on physical nodes are managed by the Virtual Machine Manager (VMM), which runs on the Operating System (OS) of physical nodes.

A cloud computing system often sets n business nodes and m redundant nodes to support fault tolerance; however, it results in high fault tolerance cost. To solve

the system-level VM fault tolerance problem, this study proposes a multi-objective optimization VM placement model based on the dual-module dynamic redundant structure. The redundant VMs are the hot spare and have been widely used in system-level VM fault tolerance. The redundant VMs and service-providing VMs are placed on different physical nodes to prevent single-point failure without providing proprietary redundant physical nodes, which can reduce the fault tolerance cost. Therefore, the two types of VM in systems are the VM that provides service and the VM that enables Replication (VMRE), which corresponds to the service-providing VM. The number of VMRE is determined on the basis of the reliability requirements of the system. As shown in Fig. 1, for example, the service-providing VM_{11} and the redundant $VMRE_{21}$ run on $node_{11}$, whereas $VMRE_{11}$ corresponding to VM_{11} and VM_{21} corresponding to $VMRE_{21}$ run on $node_{b1}$.

We assume that the system only considers the permanent or transient failure of a single VM. The system also considers the failure of the hardware and software. When a VM fails, its corresponding VMRE will replace it to provide service. Moreover, the VMM is responsible for the recovery of the resources occupied by the failed VM.

The VM fault-tolerant placement management system is built on the unified resource layer, which is composed of a monitor, a global controller (including a fault-tolerant model solver and a model manager), and a fault-tolerant strategy executor. These components form a closed-loop control system. The VMM passes run-time data, such as the information of the resource occupied by each VM and the new resource request information from the VM, to the monitor in real time. The monitor receives data from the VMM, such as the request to join the system from a new physical node or VM, calculates its SLA violation rate, resource utilization ratio, power consumption rate, and fault tolerance cost, and passes the data to the model manager in the global controller. The model manager receives the data from the monitor, processes them on the basis of the format required by the fault-tolerant model solver, and passes them to the model solver. The model solver implements the replacement strategy on the basis of the VM fault-tolerant placement model and passes it to the fault-tolerant strategy executor. The fault-tolerant strategy executor parses the placement strategy into VMM executable instructions and passes them to the VMM on the physical node where the VM needs to be replaced.

VM dynamic management is based on the structure of the initial placement, and this initial structure will affect the efficiency of VM dynamic management. This study investigates the initial placement of VMs in data centers. When available resources change, a new VM placement solution is needed.

2.2 Mathematical description of the problem

The VM placement problem is a multidimensional packing problem, which is actually a multi-objective optimization problem. To address this problem, we need to consider not only the resource allocation problem but also the SLA violation rate, resource remaining rate, power consumption rate, and fault tolerance cost. VM fault-tolerant placement needs to consider not only the service-providing VMs but also the redundant VMs, because the VMRE cannot be placed together with its corresponding service-providing VM to prevent single-point failure, which increases the fault tolerance cost.

To consider the previously mentioned factors synthetically, we need to establish a fault-tolerant placement model. We assume that the number of physical nodes in a data center is s, the number of VMs

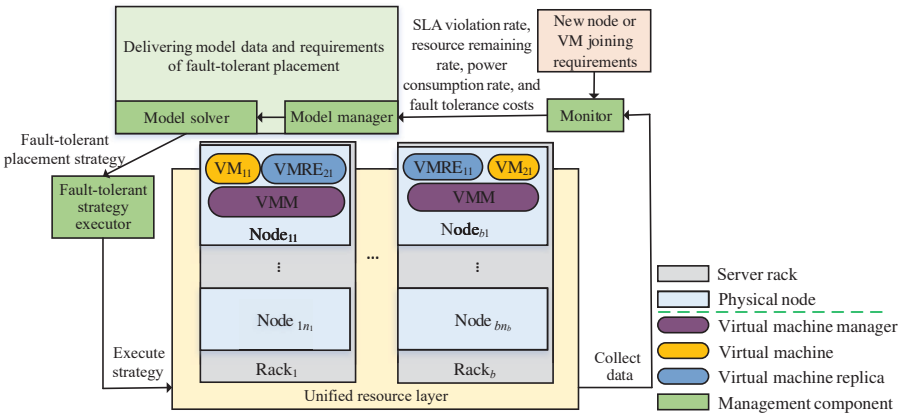


Fig. 1 Structure of the unified resource layer and fault-tolerant placement management system.

is v, and the fault-tolerant placement of VMs needs to consider the following factors:

(1) SLA violation rate

The SLA violation rate is inversely proportional to the quality of service provided by the cloud computing system. The Million Instructions Per Second (MIPS) requested by all VMs on the i-th physical node is M_{ri}, whereas the actual MIPS provided by the i-th physical node is M_{ai}. Thus, the SLA violation rate for the i-th physical node can be expressed as Eq. (1)[17,23]:

$$S_i = \frac{M_{ri} - M_{ai}}{M_{ri}}, i = 1, 2, \ldots, s \tag{1}$$

(2) Resource remaining rate

VM placement involves multiple types of resources, such as CPU, memory, disk, and network bandwidth. The balanced use of various resources is important[22]. If one type of resource is exploited, then other VMs can no longer be placed on the physical node, which will result in the waste of other resources. Figure 2 shows the resource usage example when three VMs are placed on a physical node. The horizontal axis represents memory usage and the longitudinal axis represents CPU usage. As shown in Fig. 2, the CPU usage is low (i.e., the CPU resource remains large). However, the remaining memory is inadequate for more VMs, which will result in considerable waste of physical resources.

From the previously presented analysis, we determine that VMs should be placed to minimize the resource remaining rate. The number of resource types is set to l. R_j^i represents the ratio of the remaining resource j of physical node i to its total resources, $j = 1, \ldots, l$, and $\overline{R}^i$ represents the average remaining rate of various types of resources on physical node i, $\overline{R}^i = \sum_{j=1}^{l} R_j^i \Big/ l$. Then, we define the resource remaining rate RE_i for physical node i, which is expressed as Eq. (2):

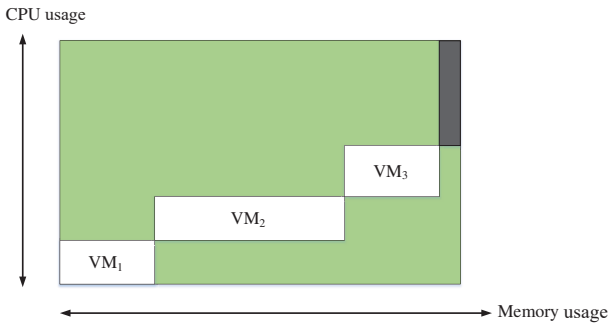


Fig. 2 Diagram of resource usage when three VMs are placed on a physical node.

$$\mathrm{RE}_i = \sqrt{\frac{\sum_{j=1}^{l}(R_j^i - \overline{R}^i)^2}{l}} \tag{2}$$

As shown in Eq. (2), if the resources are used in a balanced manner, then the value of RE_i will be relatively small.

(3) Power consumption rate

The power consumption of a data center is mainly generated by the operation of a physical machine. In Ref. [19], the authors indicated that the CPU utilization ratio was the main factor that affected the power consumption of the physical machine. Moreover, the power consumption of the physical machine increases linearly when the CPU utilization ratio increases from 0 to 1. We assume that the CPU utilization ratios of 0 and 1 correspond to the power consumption rates $P_{\min}$ and $P_{\max}$, respectively, and the CPU utilization ratio of physical machine i is u_i^{cpu}. Then, its power consumption rate P_i can be expressed as Eq. (3)[22]:

$$P_i = P_{\min} + (P_{\max} - P_{\min}) \times u_i^{\mathrm{cpu}} \tag{3}$$

(4) Fault tolerance cost

Generally, only a certain number of components are the key components in a cloud computing system[30]. We assume that the number of service-providing VMs is v, some of them are critical, and their proportion is u. Then, the number of VMs that need to be placed in a redundant manner is $r = \lfloor v \times u \rfloor$. Therefore, the data center has $v + r$ VMs, and r redundant VMs will be placed on t physical nodes. When redundant VMs are placed on a physical node, the SLA violation rate, resource remaining rate, and power consumption rate of that node will change with the amount of resource requested by the redundant VMs. Therefore, the fault tolerance cost of physical node i (noted as M_i) can be defined as a product of the proportion of the resources of a physical node consumed by redundant VMs, which can be expressed as Eq. (4):

$$M_i = \sum_{k \in S} us_{ik}^{\mathrm{cpu}} \times us_{ik}^{\mathrm{mem}} \times us_{ik}^{\mathrm{band}} \tag{4}$$

where $S = \{k | k > v \bigwedge a_{ki} = 1\}$; us_{ik}^{cpu}, us_{ik}^{mem}, and us_{ik}^{band} represent the resource proportion of the CPU, memory, and network bandwidth placed on physical node i by redundant VMs, respectively. If the k-th VM ($k = 1, \ldots, v + r$) is placed on physical node i, then $a_{ki} = 1$; otherwise, $a_{ki} = 0$.

(5) Failure rate

We assume that cloud applications consist of multiple distributed components, each of which is deployed on

a VM. According to the description in Eq. (4), this study only considers the failure of key components and implements redundant placement of VMs with key components. We assume that, corresponding to the nu key components $\mathrm{Com}_1, \ldots, \mathrm{Com}_{nu}$, the failure ratio is $f_1, \ldots, f_{nu}$. Given the use of the dual-module dynamic redundant structure, the failure ratio of the system is expressed as Eq. (5):

$$f = \sum_{y=1}^{nu} f_y^2 \tag{5}$$

We assume that the amount of CPU, memory, and network bandwidth resources possessed by physical node i can be expressed as $C_i = [C_i^{\mathrm{cpu}}, C_i^{\mathrm{mem}}, C_i^{\mathrm{band}}]$, the amount of CPU, memory, and network bandwidth resources requested by the k-th VM can be expressed as $Q_k = [Q_k^{\mathrm{cpu}}, Q_k^{\mathrm{mem}}, Q_k^{\mathrm{band}}]$, the number of physical nodes placed by redundant VM is t, the resource remaining rate of physical node i is RE_i, the VM set is C, and the redundant VM set is C_{red}. Moreover, $v + r$ VMs are placed on s physical nodes, with the placement matrix defined as A.

VM fault-tolerant placement aims to place v VMs and r redundant VMs on s physical nodes. The limiting condition is that the amount of resources requested by the VM cannot exceed the amount that the physical nodes can supply. Each VM can only be placed on one physical node. Meanwhile, a VM that requires fault-tolerant placement and its corresponding redundant VM cannot be placed on the same physical node. The optimization goal is to minimize the SLA violation rate, resource remaining rate, power consumption rate, and fault tolerance cost. Therefore, the formal description of the multi-objective optimization problem of VM fault-tolerant placement is expressed as follows:

Target:

$$\min \left[\sum_{i=1}^{s-t} S_i, \sum_{i=1}^{s-t} \mathrm{RE}_i, \sum_{i=1}^{s-t} P_i, \sum_{i=1}^{t} M_i \right] \tag{6}$$

Constraint conditions:

$$\sum_{k=1}^{v+r} Q_k^{\mathrm{cpu}} \times a_{ki} < C_i^{\mathrm{cpu}} \tag{7}$$

$$\sum_{k=1}^{v+r} Q_k^{\mathrm{mem}} \times a_{ki} < C_i^{\mathrm{mem}} \tag{8}$$

$$\sum_{k=1}^{v+r} Q_k^{\mathrm{band}} \times a_{ki} < C_i^{\mathrm{band}} \tag{9}$$

$$\sum_{i=1}^{s} a_{ki} = 1 \tag{10}$$

$$a_{ki} + a_{(v+k)i} \leqslant 1, k = 1, \ldots, r \tag{11}$$

Formula (6) is an objective function of multi-objective optimization. Formulae (7) – (9) represent the total amount of CPU, memory, and network bandwidth resources requested by VMs, respectively, which cannot exceed the amount of resources actually possessed by the physical node. Equation (10) indicates that each VM can only be placed on one physical node. To facilitate the design of the model and algorithm, we set the key service-providing r VMs from the first to r-th and set their corresponding r redundant VMs from $(v + 1)$-th to $(v + r)$-th. Formula (11) indicates that the r key VMs and their r corresponding redundant VMs cannot be placed on the same physical node.

3 Multi-Objective Optimization Algorithm of Initial VM Fault-Tolerant Placement

Generally, we can obtain the optimal solution for single-objective optimization using several algorithms. However, obtaining such optimal solution for multi-objective optimization is difficult. The multi-objective optimization problem has several Pareto solutions, and we often select one as the optimal solution[32].

Heuristic and evolutionary algorithms are often used to solve the VM placement problem. The four types of traditional heuristic algorithms are the first fit, best fit, First Fit Descending (FFD), and Best Fit Descending (BFD) algorithms[15]. The two types of evolutionary algorithms are genetic[13] and ant colony[15,23] algorithms. The Ant Colony Optimization (ACO) algorithm is a bionic optimization algorithm, which has strong robustness[33] and can use the positive feedback mechanism to obtain the optimal approximate solution of NP combinatorial optimization problem rapidly.

VM fault-tolerant placement is a discrete multi-objective optimization problem, which should consider not only the placement of service-providing VMs but also the placement of redundant VMs. Moreover, the service-providing VM cannot be placed on the same physical node with its corresponding redundant VM. This study proposes two heuristic ACO algorithms to solve the multi-objective optimization problem of VM fault-tolerant placement for data centers of cloud systems, i.e., First Fit Descending Ant Colony Optimization (FFDACO) and Best Fit Descending Ant Colony Optimization (BFDACO) algorithms.

3.1 Fitness function

According to Formula (6), the multi-objective

optimization model needs to minimize the values of the four factors collaboratively. To optimize the four objective functions, the objective functions of the SLA violation rate, resource remaining rate, power consumption rate, and fault tolerance cost are derived as follows:

(1) Function of the SLA violation rate

According to Eq. (1), S_i is related to the CPU utilization ratio of physical node i, and the higher the CPU utilization ratio, the larger the value of S_i. To make the value change in the range of $(0, 1)$[23], the function of the SLA violation rate is expressed as Eq. (12):

$$F_{\text{SLA}}(u_i^{\text{cpu}}) = \frac{1}{1 + \text{e}^{u_i^{\text{cpu}} - 0.9}} \tag{12}$$

where u_i^{cpu} represents the CPU utilization ratio of physical node i. The experimental results presented in Ref. [23] indicate that, when the CPU utilization ratio increases from 0 to 0.9, the SLA violation rate increases gradually. However, when the CPU utilization ratio is > 0.9, the SLA violation rate increases rapidly. Therefore, the threshold of the CPU utilization ratio is set to 0.9. Our experimental environment is similar to that in Ref. [23]. Thus, we also set the threshold to 0.9.

(2) Function of the resource remaining rate

According to Eq. (2), the function of the resource remaining rate of physical node i can be expressed as Eq. (13):

$$F_{\text{res}}(U_i) = (1 - u_i^{\text{cpu}}) \times (1 - u_i^{\text{mem}}) \times (1 - u_i^{\text{band}}) \tag{13}$$

where $U_i = (u_i^{\text{cpu}}, u_i^{\text{mem}}, u_i^{\text{band}})$, u_i^{mem} is the memory utilization ratio and u_i^{band} is the network bandwidth utilization ratio of the physical node i. Moreover, $u_i^{\text{cpu}} = \sum_{k=1}^{v} a_{ki} \times Q_k^{\text{cpu}} \Big/ C_i^{\text{cpu}}$, $u_i^{\text{mem}} = \sum_{k=1}^{v} a_{ki} \times Q_k^{\text{mem}} \Big/ C_i^{\text{mem}}$, and $u_i^{\text{band}} = \sum_{k=1}^{v} a_{ki} \times Q_k^{\text{band}} \Big/ C_i^{\text{band}}$.

As shown in Eq. (13), the higher the resource utilization ratio, the lower the resource remaining rate.

(3) Function of the power consumption rate

According to Eq. (3), P_i and u_i^{cpu} are linearly related. Given that the values of the SLA violation rate and the resource remaining rate are in the range of [0, 1], we also make the value of P_i change in the range of [0, 1]. The function of the power consumption rate of physical node i is expressed as Eq. (14):

$$F_{\text{power}}(U_i^{\text{cpu}}) = \frac{u_i^{\text{cpu}}}{P_{\min} + (P_{\max} - P_{\min}) \times u_i^{\text{cpu}}} \times P_{\max} \tag{14}$$

As shown in Eq. (14), the function of the power consumption rate is a monotonous increment function, and the larger the value of u_i^{cpu}, the higher the power consumption rate. Therefore, we need to reduce its value as much as possible.

(4) Function of the fault tolerance cost

On the basis of Eq. (4), the function of the fault tolerance cost of physical node i is expressed as Eq. (15). Notably, the fault tolerance cost is affected by the resource requests from redundant VMs.

$$F_{\text{ft}}^i = \frac{\sum_{k=v+1}^{v+r} a_{ki} \times Q_k^{\text{cpu}}}{C_i^{\text{cpu}}} \times \frac{\sum_{k=v+1}^{v+r} a_{ki} \times Q_k^{\text{mem}}}{C_i^{\text{mem}}} \times \frac{\sum_{k=v+1}^{v+r} a_{ki} \times Q_k^{\text{band}}}{C_i^{\text{band}}} \tag{15}$$

Given that the values of Eqs. (12) – (15) are all in the range of [0, 1], on the basis of Formula (6), the fitness function of the multi-objective optimization of the VM fault-tolerant placement model can be expressed as Eq. (16):

$$\begin{aligned} F(U) = {} & w_1 \times \sum_{i=1}^{s-t} F_{\text{SLA}}(u_i^{\text{cpu}}) + \\ & w_2 \times \sum_{i=1}^{s-t} F_{\text{res}}(U_i) + \\ & w_3 \times \sum_{i=1}^{s-t} F_{\text{power}}(u_i^{\text{cpu}}) + \\ & w_4 \times \sum_{i=1}^{s} F_{\text{ft}}^i \end{aligned} \tag{16}$$

where $w_1 + w_2 + w_3 + w_4 = 1$, w_1, w_2, w_3, and w_4 represent the weights of the SLA violation rate, resource remaining rate, power consumption rate, and fault tolerance cost, respectively, and U is the matrix constructed using U_i.

3.2 Heuristic ant colony algorithm

The following part describes the traditional heuristic

algorithm, which can solve the VM fault-tolerant placement problem, and proposes a heuristic ACO algorithm.

(1) FFD-based algorithm of VM fault-tolerant placement

The FFD-based algorithm of VM fault-tolerant placement is described in Algorithm 1. Its input contains C_i, Q_k, u, $P_{\min}$, $P_{\max}$, w_1, w_2, w_3, and w_4, and its output contains A and $F(U)$. A is the placement matrix that describes how the VMs are placed on physical nodes. This algorithm can be divided into three steps, as follows:

Step 1. Sort the resource request of VMs in descending order, expressed as $a_1, a_2, \ldots, a_{v+r}$.

Step 2. Place a_1 on the physical node hn_1. If hn_1 does not meet the requirements, then select the next physical node hn_2, until the resource request of a_1 is met.

Algorithm 1. FFD-based VM fault-tolerant placement

Input: C_i, Q_k, u, $P_{\min}$, $P_{\max}$, w_1, w_2, w_3, w_4;
Output: A: placement matrix, $F(U)$: value of fitness function;

```
Initialize A, U;
Sort the resource request from VMs in descending order, expressed as a_1,a_2,...,a_{v+r};
i = 1;
while a_1 not been placed do
    if physical node hn_i meets the requirements of a_1 then
        initialize physical node hn_i;
        place a_1 on the physical node hn_i, update A;
    i = i + 1;
j = 2;
while j ⩽ v + r do
    mark initialized physical nodes as hn*_1, hn*_2, ..., hn*_z (keep original order);
    g = 1;
    while a_j not been placed and g ⩽ z do
        if hn*_g meets the requirements of a_j and placing a_j on hn*_g will not break the rule defined by Eq. (10) then
            place a_j on the physical node hn*_g, update A;
        g = g + 1;
    if a_j not been placed then
        initialize a new physical node and place a_j on it, update A;
    j = j + 1;
calculate U according to A and F(U) according to Eq. (16), respectively;
return A, F(U);
```

Step 3. To meet the resource request a_k of the VM, if physical nodes $\text{hn}_1^*, \text{hn}_2^*, \ldots, \text{hn}_z^*$ are already in use at this time, then select one or some of them to place a_k in numerical order of subscript, ensuring that the resource request is met and its corresponding redundant and service-providing VMs are not on the same physical node. If one or some physical nodes that meet the requirements do exist, then select the one that has the minimum subscript to place a_k. By contrast, if such a physical node does not exist, then select a new physical node to place a_k.

(2) BFD-based algorithm of VM fault-tolerant placement

The BFD-based algorithm of VM fault-tolerant placement is described in Algorithm 2. Its input and output are the same as those of the FFD-based algorithm. This algorithm can be divided into three steps, as follows:

Step 1. Sort the resource requests of VMs in descending order, expressed as $a_1, a_2, \ldots, a_{v+r}$.

Step 2. Place a_1 on the physical node hn_1. If hn_1 does not meet the requirements, then select the next physical node hn_2, until the resource request of a_1 is met.

Step 3. To meet the resource request a_k of the VM, if physical nodes $\text{hn}_1^*, \text{hn}_2^*, \ldots, \text{hn}_z^*$ are already in use at this time, then select one or some of them to place a_k in numerical order of the subscript, ensuring that the resource request is met and its corresponding redundant and service-providing VMs are not on the same physical node. If one or some physical nodes that meet the requirements do exist, then select the one which can minimize the resource remaining rate to place a_k. By contrast, if such a physical node does not exist, then select a new physical node to place a_k.

The traditional ant colony algorithm optimization depends on the positive feedback mechanism, but it often leads to local optimization or premature convergence. To avoid this situation, this study uses the Max-Min Ant System (MMAS) proposed by Stützle and Hoos[33], which can search for high-quality solutions and avoid premature convergence or falling into the local optimum.

We assume that the number of ants is n in iteration q, the number of ants on the k-th VM is $b_k(q)$, the pheromone of the k-th VM on the physical node i is $\tau_{ki}(q)$, the probability that ant h places the k-th VM on physical node i is $P_{ki}^h(q)$, the heuristic function is $\eta_{ki}(q)$, the information heuristic factor is ∂, the expected heuristic factor is β, the tabu list of ant h is $\text{tabu}_h(q)$ (the

Algorithm 2. BFD-based VM fault-tolerant placement

Input: C_i, Q_k, u, $P_{\min}$, $P_{\max}$, w_1, w_2, w_3, w_4;
Output: A: placement matrix, $F(U)$: value of fitness function;

```
Initialize A, U;
Sort the resource request from VMs in descending order, expressed as a1,a2,...,av+r;
i = 1;
while a1 not been placed do
    if physical node hn_i meets the requirements of a1 then
        initialize physical node hn_i;
        place a1 on the physical node hn_i, update A;
    i = i + 1;
j = 2;
while j ⩽ v + r do
    mark initialized physical nodes as hn*_1, hn*_2, ..., hn*_z;
    g = 1, pos = −1, minRes= 1;
    while g ⩽ z do
        if hn*_g meets the requirements of a_j and placing a_j on hn*_g will not break the rule defined by Eq. (10) then
            assuming that a_j is placed on the physical node hn*_g, calculate the resource remaining rate according to Eq. (13), assigning its value to tmp;
            if tmp < minRes then
                pos = g;
                minRes = tmp;
        g = g + 1;
    if pos != −1 then
        place a_j on physical node hn*_pos, update A ; if
    a_j not been placed then
        initialize a new physical node and place a_j on it, update A;
    j = j + 1;
calculate U according to A and F(U) according to Eq. (16), respectively;
return A, F(U);
```

set of VMs that has been placed), the volatile coefficient $\rho \in [0, 1)$, the pheromone increment of the k-th VM on physical node i is $\Delta\tau_{ki}^{\text{best}}$, and the set of VMs that has not yet been placed and can be placed on physical node i by ant h is allowed$_h$. If $k \in$ allowed$_h$, then the transition probability function $P_{ki}^h(q)$ can be expressed as Eq. (17); otherwise, its value is zero.

$$P_{ki}^h(q) = \frac{\tau_{ki}(q)^{\partial} \times \eta_{ki}(q)^{\beta}}{\sum_{k \in \text{allowed}_h} (\tau_{ki}(q)^{\partial} \times \eta_{ki}(q)^{\beta})} \tag{17}$$

where ∂ is a parameter used to control the influence of the information accumulated by ants during its movement process and β is a parameter used to control the influence of the heuristic information in path selection. In iteration q, the pheromone function of the k-th VM on physical node i is expressed as Eq. (18):

$$\tau_{ki}(q+1) = (1-\rho) \times \tau_{ki}(q) + \Delta\tau_{ki}^{\text{best}} \tag{18}$$

In the first iteration, $\tau_{ki}(0) = \alpha$, α is a constant value and $1 - \rho$ is the pheromone evaporation coefficient. In MMAS, we need to set the upper and lower bounds for pheromone. We assume that $\tau_{ki}(q) \in [\tau_{\min}, \tau_{\max}]$, where the initial value of $\tau_{\max}$ is $\tau_{ki}(0)$, $\tau_{\min} = \dfrac{\tau_{\max}}{g}$, and g is the lower bound factor (i.e., $g > 1$). The pheromone increment $\Delta\tau_{ki}^{\text{best}}$ of the k-th VM on the physical node i is expressed as Eq. (19):

$$\Delta\tau_{ki}^{\text{best}} = \begin{cases} \dfrac{F(A^{\text{best}})}{\tau_{ki}(q)}, & \text{if } \alpha_{ki} = 1; \\ 0, & \text{else} \end{cases} \tag{19}$$

where A^{best} represents the optimal solution set of this model. According to Eq. (19), the value of $F(A^{\text{best}})$ can be calculated. In MMAS, the pheromone increment is equal to the value of the fitness function in the optimal solution. However, in this algorithm, the pheromone increment is equal to the value of the fitness function in the optimal solution divided by the pheromone, which can avoid falling into the local optimum prematurely and obtain the global optimal approximate solution set eventually. In Eq. (17), $\eta_{ki}(q)$ is the heuristic function when the k-th VM is placed on physical node i and represents the desirability of the k-th VM placed on physical node i. Its function can be expressed as Eq. (20):

$$\eta_{ki}(q) = \frac{1}{1 - Q_k^{\text{cpu}'}} \times \frac{1}{1 - Q_k^{\text{mem}'}} \times \frac{1}{1 - Q_k^{\text{band}'}} \tag{20}$$

where $Q_k^{\text{cpu}'}$, $Q_k^{\text{mem}'}$, and $Q_k^{\text{band}'}$ represent the ratios of the request of the k-th VM for CPU, memory, and network bandwidth resources to the remaining resources of physical node i, respectively. As shown in Eq. (20), the ant preferentially places VMs with large requests for CPU, memory, and network bandwidth resources.

3.3 Multi-objective optimization algorithm of initial VM fault-tolerant placement

The heuristic ant colony algorithm includes the FFDACO and BFDACO algorithms. The difference is that, when an ant has placed all of the service-providing VMs, the redundant VMs can be placed by the FFD or BFD algorithm.

The FFDACO-based algorithm of VM fault-tolerant placement is described in Algorithm 3. Its input and output are similar to those of Algorithms 1 and 2. The difference is that the FFDACO-based algorithm has several additional parameters to MMAS that we have discussed in Section 3.2. This algorithm can be divided into the following steps:

Step 1. Initialize the parameters, i.e., the number of ants is N_{ant} and the number of iterations is N_{run}.

Step 2. Select the physical node i randomly and place ant h on it. Then, the ant starts to search VMs.

Step 3. If set C (set of service-providing VMs) is not empty, then proceed to Step 4; otherwise, proceed to Step 6.

Step 4. The ant h searches the VMs from allowed$_h$, which contains the VMs that have not been placed and can be placed on physical node i. If set allowed$_h$ is empty, then select the next physical node randomly and proceed to Step 3.

Step 5. The ant h updates $\eta_{ki}(q)$ (it represents the desirability of the k-th VM placed on physical node i) according to Eq. (20), selects the k-th VM from set allowed$_h$, places it on the physical node i using the roulette wheel selection algorithm (which selects randomly by probability) according to Eq. (17), updates the placement matrix A, deletes the k-th VM from set C, and updates C_i (useable resources on the physical node i). Then, proceed to Step 3.

Step 6. Place the redundant VMs. Sort the VMs in set C_{red} on the basis of the resource requests in descending order. Then, new VM sequence can be expressed as $\text{VM}_1, \text{VM}_2, \ldots, \text{VM}_r$.

Step 7. If set C_{red} is not empty, then proceed to Step 8; otherwise, proceed to Step 9.

Step 8. To meet the resource request of VM_k in set C_{red}, if physical nodes hn_1^*, hn_2^*, ..., hn_z^* are already in use at this time, then select one or some of them to place the VM_k in numerical order of subscript, ensuring that the resource request is met and its corresponding redundant and service-providing VMs are not on the same physical node. Then, if one or some physical nodes

Algorithm 3. FFDACO-based VM fault-tolerant placement

Input: C_i, Q_k, u, $P_{\min}$, $P_{\max}$, w_1, w_2, w_3, w_4, ∂, β, ρ, g, $\tau_{ki}(0)$, N_{ant}, N_{run};
Output: A^{best}: best placement matrix, $F(U)$: value of fitness function;

```
Initialize A, U, A^best, C, C_red;
h = 1, q = 1;
while q ⩽ N_run do
    while h ⩽ N_ant do
        ant h randomly selects physical node i;
        while C is not empty do
            if allowed_h is not empty then
                update η_ki(q) according to Eq. (20);
                select VM_k from C by roulette wheel
                  selection algorithm according to Eq. (17);
                place VM_k on physical node i, update A;
                delete VM_k from C;
                update C_i;
            else
                ant h randomly selects physical node i;
        Sort VMs in set C_red by resource request in
          descending order, the new VM sequence
          expressed as VM_1,VM_2,...,VM_r;
        k = 1;
        while k ⩽ r do
            mark initialized physical nodes as
            hn_1^*, hn_2^*, ..., hn_z^* (keep original order);
            g = 1;
            while g ⩽ z do
                if hn_g^* meets the requirements of VM_k
                  and placing VM_k on hn_g^* will not break
                  the rule defined by Eq. (10) Then
                    place VM_k on the physical node hn_g^*,
                      update A;
                  g = g + 1;
            if VM_k not been placed then
                initialize a new physical node and place
                  VM_k on it, update A;
            k = k + 1;
        if h == 0 and q == 0 then
            A^best = A;
        else
            if (F(A) < F(A^best)) then
                A^best = A;
        reset C_i, A, C, η_ki(q);
        h = h + 1;
    update τ_ki(q) according to Eqs. (18) and (19);
    if τ_ki(q) < τ_min then
        τ_ki(q) = τ_min;
    if τ_ki(q) > τ_max hen
        τ_ki(q) = τ_max;
    q = q + 1, h = 0;
calculate U according to A^best and F(U) according to
  Eq. (16), respectively;
return A^best, F(U);
```

that meet the requirements do exist, then select the one that has the minimum subscript to place the VM_k. If such physical node does not exist, then select a new physical node to place the VM_k. Finally, delete the VM_k from C_{red}, update the placement matrix A, and proceed to Step 7.

Step 9. If $h = 0$, then $A^{best} = A$; otherwise, compare $F(A)$ with $F(A^{best})$, if $(F(A) < F(A^{best})$, then $A^{best} = A$. Reset C_i, $A = 0$, C, and $\eta_{ki}(q)$. Then set $h = h + 1$, if $h < N_{ant}$, proceed to Step 2.

Step 10. Update $\tau_{ki}(q)$ according to Eqs. (18) and (19), if $\tau_{ki}(q) < \tau_{min}$, set $\tau_{ki}(q) = \tau_{min}$; if $\tau_{ki}(q) > \tau_{max}$, set $\tau_{ki}(q) = \tau_{max}$. Then set $q = q+1$, if $q < N_{run}$, reset $h = 0$, and proceed to Step 2.

Step 11. Calculate U according to A^{best} and $F(U)$ by Eq. (16). Then, output A^{best} and $F(U)$.

The difference between BFDACO and FFDACO algorithms is determined in Steps 6 – 8. In the BFDACO algorithm, these steps adopt the BFD method described in Algorithm 2.

According to Ref. [33], the time complexity of the proposed heuristic ant colony algorithm is $T(n) = O(q\times (v+r)^2 \times n)$, which increases with the increment of the number of VMs $(v + r)$, the number of ants (n), and the number of iterations (q), respectively. Its space complexity is $S(n)=O((v + r)^2)+ O((v + r) \times n)$; therefore, it increases with the increment of the number of VMs and the number of ants.

The four algorithms (i.e., FFD, BFD, FFDACO, and BFDACO) are all implemented using the Java programming language. Section 4 describes the three experiments, i.e., simulation, real cluster, and fault injection experiments, conducted in this study and explains all of the related algorithmic languages and environments.

4 Experiment and Analysis

To verify the VM fault-tolerant placement method proposed in this study, the multi-objective optimization model is realized in a simulation environment. Then, the results of the simulation experiment are verified in a real cluster environment. Finally, the failure rate of the system is evaluated by the fault injection experiment in a real cluster environment.

4.1 Simulation experiment and analysis

The simulation experiment is realized on the CloudSim platform, which is a cloud computing platform simulation software developed by the Grid Laboratory of Melbourne University and the Gridbus Project[34]. The CloudSim platform provides a function library based on the discrete event simulation package SimJava and supports multiple Operating Systems (OSs).

In the simulation experiment, the CloudSim platform holds 110 physical nodes. The processing capability of the CPU is divided into three levels, i.e., 1001, 2001, and 3001 MIPS, and each physical node has 10 001 MB memory, 1001 GB disk, and 1001 Mbps network bandwidth. We need to deploy a total of 250 service-providing VMs, and their request for CPU resource can be divided into four types, i.e., 250, 500, 750, and 1000 MIPS, in which each VM requests for 250 MB memory, 200 GB disk and 250 Mbps network bandwidth. When the CPU utilization ratio is 0, the power consumption rate is $P_{min} = 75$ W. When the CPU utilization ratio increases to 1, the power consumption rate is $P_{max} = 175$ W. When $u = 0.2$, the number of redundant VMs is 50 $(r = 50)$. In Eq. (16), w_1, w_2, w_3, and w_4 are the weights of different placement factors, which represent the influence of different factors, and can be set manually in simulation experiments and configured by the system administrator in real systems. Because, in different applications and environments, the requirements of users are different. For example, if users pay more attention to service quality, then w_1 can be set to have a large value but should be in the range [0, 1]. If users only care about service quality, then w_1 can be set to 1, $w_2 = w_3 = w_4 = 0$, and the problem degrades to a single-constraint optimization problem.

We realized four kinds of VM fault-tolerant placement algorithms, i.e., FFD, BFD, FFDACO, and BFDACO algorithms, in the simulation experiments. Given that the amounts of resources requested for memory, disk, and network bandwidth are the same, when we use the heuristic algorithm or heuristic ant colony algorithm, the requested resource will be sorted on the basis of the processing capability of the CPU.

The parameters of heuristic ant colony algorithm are determined by experimental training, in which ∂, β, and g are integers (i.e., $\partial \in [1, 10]$, $\beta \in [1, 10]$, and $g \in [1, 10]$), and the increment of ρ is 0.01 and $\rho \in (0, 1)$. When the number of ants is > 15 and the number of iterations is > 200, the variation of the value of the fitness function is small. Therefore, we set $\partial = 2$, $\beta = 5$, $\rho = 0.3$, $g = 4$, $\tau_{ki}(0) = 3$, $N_{ant} = 15$, and $N_{run} = 200$.

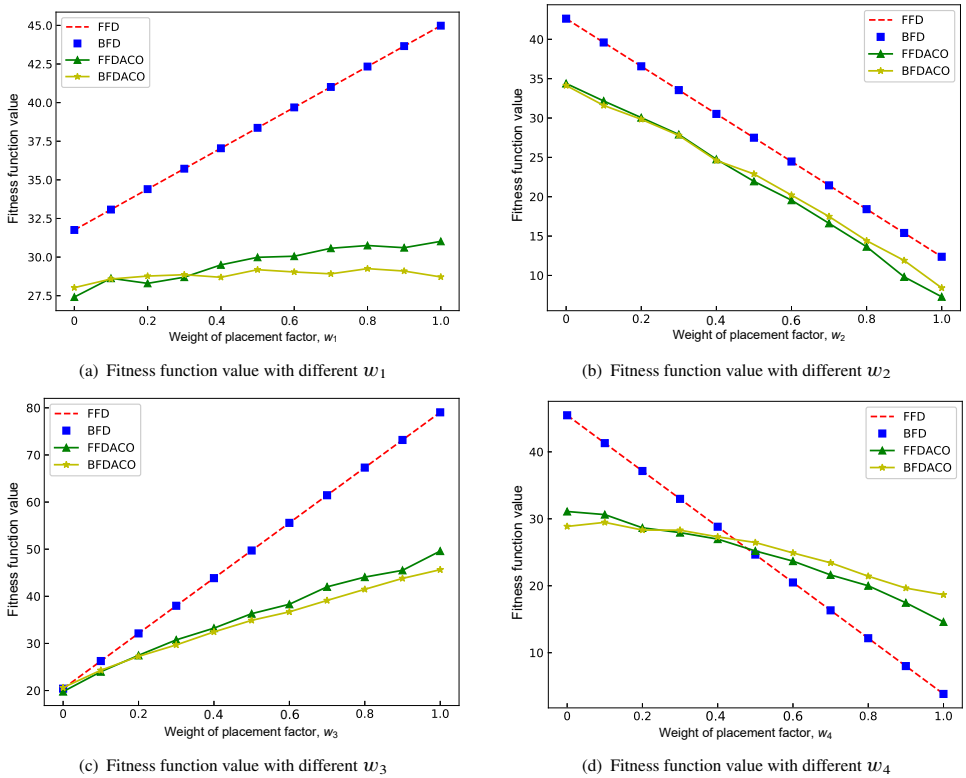
For each $w_z(z \in \{1, 2, 3, 4\})$, the value is initially set from 0 to 1 (recorded as p) and subsequently set to

$(1-p)/3$. The variations of the fitness function values based on the FFD, BFD, FFDACO, and BFDACO are shown in Fig. 3. As shown in Figs.3a and 3c, the fitness function value increases with the increase in w_1 (weight of the SLA violation rate) and w_3 (weight of the resource remaining rate). Moreover, the fitness function values based on the FFDACO and BFDACO are lower than those based on the FFD and BFD. When w_1 and w_3 are > 0.3, BFDACO has a lower fitness function value than FFDACO. As shown in Figs. 3b and 3d, the fitness function value decreases with the increase in w_2 (weight of the power consumption rate) and w_4 (weight of the fault tolerance cost). In Fig. 3b, the fitness function values based on the FFDACO and BFDACO are lower than those based on the FFD and BFD. Moreover, when w_2 is > 0.4, FFDACO has a lower fitness function value than BFDACO. In Fig. 3d, when $w_4 < 0.2$, BFDACO has a low fitness function value; when $0.2 < w_4 < 0.5$, FFDACO has a low fitness function value; and when w_4 is > 0.5, FFD and BFD have low fitness function values.

In the next part of the simulation and real cluster experiments, we assume that $w_1 = w_2 = w_3 = w_4 = 0.25$. The fitness function values based on the FFD, BFD, FFDACO, and BFDACO are 34.053 78, 34.053 78, 29.702 65, and 29.466 14, respectively. The results indicate that the fitness function value of the algorithm based on the FFDACO or BFDACO is less than that of the algorithm based on the FFD or BFD. Moreover, the heuristic ant colony algorithm proposed in this study outperforms the heuristic algorithm in solving the fault-tolerant placement problem of VMs. This finding can be attributed to the fact that the heuristic ant colony algorithm utilizes the positive feedback mechanism and the characteristics of MMAS, which can optimize the multiple factors of VM fault-tolerant placement and search for the global optimal solution. Thus, the fitness function value can reach the optimal value. No difference in fitness function value between FFD-based and BFD-based algorithms can be observed because they all use the physical nodes that have already been used firstly.

Figure 4 shows the comparison of the values of the placement factors among the algorithms based on the FFD, BFD, FFDACO, and BFDACO.

As shown in Fig. 4, the algorithms based on the FFD and BFD have high power consumption rates, whereas the algorithms based on the FFDACO and BFDACO have high fault tolerance costs. For the algorithms based on the FFD and BFD, VMs are sorted on the basis of the amount of resource requested for CPU. Then, the physical node is selected to place the service-providing and redundant VMs together. However, for



(a) Fitness function value with different w_1

(b) Fitness function value with different w_2

(c) Fitness function value with different w_3

(d) Fitness function value with different w_4

Fig. 3 Variations of the fitness function values with different weights of placement factors.

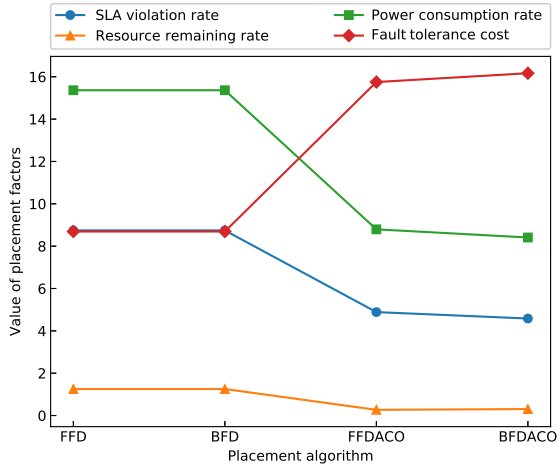


Fig. 4 Comparison of the values of the placement factors among different algorithms.

the algorithms based on the FFDACO and BFDACO, we place the service-providing VMs using the ant colony algorithm. Then, we place the redundant VMs using the FFD or BFD algorithm. The number of physical nodes required by the FFD and BFD is 105, whereas that required by the FFDACO and BFDACO is 109. If the resource requests are the same, then the number of physical nodes will be small, the resource remaining rate will be low, and the CPU utilization ratio will be high. Thus, the power consumption rates of the algorithms based on the FFD and BFD are high. In the FFDACO-based and BFDACO-based algorithms, redundant VMs are placed as far as possible on the physical nodes that are already used, leading to a high fault tolerance cost. Figures 5 and 6 show the variations of the fitness function values based on the FFDACO and BFDACO algorithms as the number of iterations increases, respectively. Notably, the algorithm based on the FFDACO or BFDACO will converge to

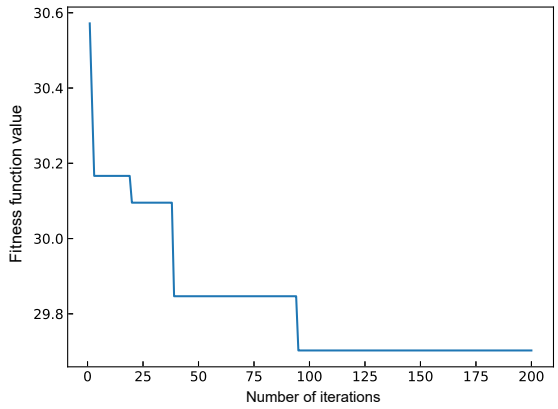


Fig. 5 Variation of the fitness function value based on the FFDACO algorithm.

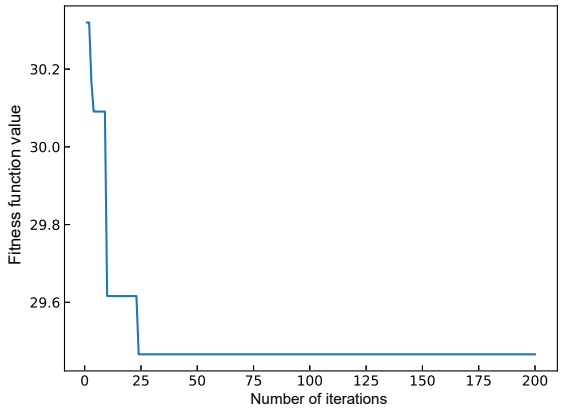


Fig. 6 Variation of the fitness function value based on the BFDACO algorithm.

a stable value after 100 iterations, because when the pheromone function is continually updated, the ants will approach the optimal placement solution. Moreover, the convergence speed of the fitness function value of the algorithm based on the FFDACO is slower than that of the algorithm based on the BFDACO because the redundant VMs of FFDACO are placed on the basis of the ordinal number of the physical nodes that have been already used. Furthermore, the ordinal number may be different from that of the last iteration. Meanwhile, for BFDACO, the physical node with the smallest resource remaining rate is selected. Thus, the convergence speed of the algorithm based on the FFDACO is slower than that of the algorithm based on the BFDACO.

4.2 Real cluster experiment and analysis

To verify the effect of the VM fault-tolerant placement solution on the real cloud platform, we conducted experiments in the laboratory cluster environment and verified and analyzed the given VM fault-tolerant placement solution.

To simulate real cloud computing services, we use the benchmark program Rice University Bidding System (RUBiS), which is similar to the eBay auction system. RUBiS is an open source benchmark program developed by Rice University and often used to evaluate the quality of design patterns for network applications[35]. The experimental platform is a cluster environment composed of six physical nodes, one of which is a monitoring node and the remaining five nodes are service nodes. The monitoring node is responsible for collecting the dynamic information of VMs on the service nodes, inputting the data into the model solver, and obtaining the output value of the objective function. The service nodes are used to place service-

providing and redundant VMs, and the service node numbers are defined as N_0, N_1, N_2, N_3, and N_4. The monitoring node configuration is Intel Core 2 Duo Processor E7500 (2.93 GHz), 4 GB memory, and 500 GB hard disk (7200 RPM). The service node configuration is Intel Xeon Processor E3-1225 v2 (3.2 GHz), 8 GB memory, and 1 TB hard disk (7200 RPM). All of the nodes are connected by a gigabit switch, and both monitoring and service nodes install CentOS 6.4 x86_64 and Java 1.8 update 5.

In the deployment of the MySQL cluster, the SQL node is responsible for accessing cluster data and the data node is responsible for saving cluster data, the number of which is related to the number of their replicas. Moreover, these two types of nodes provide the critical data service of RUBiS. Thus, these nodes need to be placed redundantly.

RUBiS is a typical three-tier architecture web application benchmark program. We selected Version 1.3, and the logical structure of the experimental system is shown in Fig. 7. 13 VMs based on Kernel-based Virtual Machine (KVM) were used in this experiment. The first tier used three VMs, and all were installed the Apache server (Version 2.4.9), which is responsible for receiving HTTP requests from clients and forwarding these request to the second tier. These VMs are labeled as VM_3, VM_4, and VM_5 in sequence. The second tier used five VMs, and all were installed the Apache tomcat server (Version 7.0.54), which is responsible for running RUBiS and receiving and processing the HTTP requests from the first tier. These VMs are labeled as VM_6, VM_7, VM_8, VM_9, and VM_{10}. The third tier used five VMs, and all were installed the MySQL cluster, of which one was installed the management component, one was installed the data component, one was installed the SQL component, and the remaining two were redundant VMs[36], which were set as the backup machines of the data and SQL components. The version of MySQL is 7.3.6. These VMs are labeled as VM_2, VM_0, VM_1, VM_{11}, and VM_{12}. The guest OS of KVM is Ubuntu 13.10 Server x86_64. The CPU of each tier of VMs is configured as single core and single thread. The memory configurations of the first, second and third tiers are 1500, 2000, and 2500 MB, respectively.

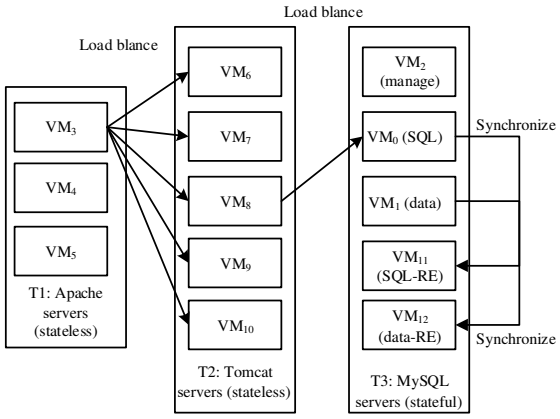


Fig. 7 Three-layer structure based on RUBiS.

The proposed VM fault-tolerant placement algorithm is used to derive the mapping solution of 13 VMs and 5 service nodes. Then, RUBiS is deployed and tested on the basis of the obtained mapping solution. The utilization ratio data of the CPU, memory, and network bandwidth of the service nodes are collected, and the actual fitness value of the objective function is calculated. Finally, the consistency of the results between real cluster and simulation experiments is verified.

To quantify the CPU processing power, the single core and single thread processing capability of the service node CPU is set to 1000 MIPS. Thus, the CPU processing capacity of each service node is 4000 MIPS and the CPU resource requested by each VM is 1000 MIPS. Given that the physical network card of the host is shared with the KVM guest OS by bridging, the network bandwidth of the service node is 100 Mbps, and the network bandwidth resource requested by each VM is 25 Mbps. To prevent the resource occupied by the VMs from being affected by the guest OS resource requirements during the experiment, the number of VMs placed on each physical node is limited to three or less.

The prototype system of the VM fault-tolerant management system presented in Fig. 1 is implemented as shown in Fig. 8.

The monitoring node of the prototype system includes the model management, solver process, and the monitoring process. The monitoring process

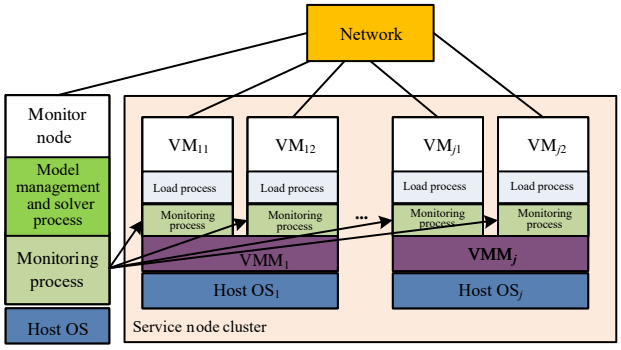


Fig. 8 Implementation of the prototype system.

is responsible for receiving resource information (including the actual utilization ratio data of CPU, memory, and network bandwidth) collected by the monitoring process of the service node cluster, forwarding the resource data to the model management and solver process, and receiving its output, i.e., the VM placement solution. Finally, the monitoring process connects to the Libvirt service on each service node through Remote Procedure Call (RPC) and performs the VM creation and placement operations on the basis of the obtained placement solution.

In the service node cluster, each VM runs the monitoring process, which is responsible for transmitting resource information to the monitoring node. The VM resource configuration on each physical node is managed by its VMM, and the operation instructions are transmitted by the interface provided by the Libvirt service.

In the real cluster experiment, the mapping solution of VMs to physical nodes is the same as that of the simulation experiment (including the settings of the model parameters). Table 1 shows the mapping solutions of the four types of VM placement algorithms. The first row shows the 13 VMs, i.e., VM_0–VM_{12} and the first column shows the four different placement algorithms. The second to fifth rows depict the mapping solutions of 13 VMs and 5 service nodes of different placement algorithms.

Figure 9 shows the fitness function values based on the four types of VM fault tolerant placement algorithms in the simulation experiment. As shown in Table 1 and Fig. 9, both VM fault-tolerant placement algorithms based on the FFD and BFD have not only the same fitness function value but also the same placement solution. However, both algorithms based on the FFDACO and BFDACO only have the same fitness function value. Their placement solutions are different because they select different VMs to place during the process of each iteration. Figure 10 shows the real cluster experimental results of the fitness function values when different VM fault-tolerant placement algorithms are used. Notably, the fitness function values of the VM fault-tolerant placement algorithms based on the FFDACO and BFDACO are smaller than those of the algorithm based on the FFD or BFD, which is consistent with the simulation results. The fitness function value of the VM fault-tolerant placement algorithm based on the BFD or BFDACO is smaller than that of the algorithm based on the FFD or FFDACO. This finding can be attributed to the fact that the algorithm based on the BFD or BFDACO prioritizes the physical node that can minimize the resource remaining rate when placing the redundant VMs.

The comparison of Figs. 9 and 10 shows that the fitness function value of the algorithm based on the FFD or BFD is larger than that of the algorithm based on the FFDACO or BFDACO. In the real cluster experiment, the value of the resource utilization ratio takes the

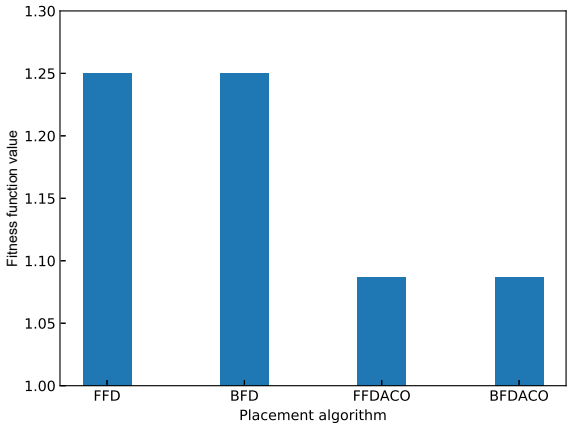


Fig. 9 Fitness function values in the simulation experiment.

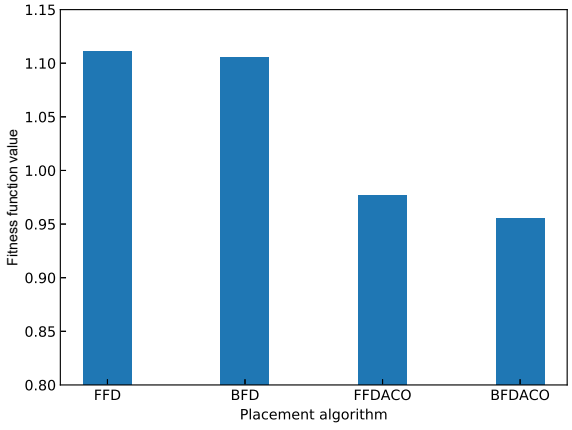


Fig. 10 Fitness function values in the real cluster experiment.

Table 1 Mapping solutions of 13 VMs and 5 service nodes.

Method	VM_0	VM_1	VM_2	VM_3	VM_4	VM_5	VM_6	VM_7	VM_8	VM_9	VM_{10}	VM_{11}	VM_{12}
FFD	N_0	N_0	N_0	N_3	N_3	N_4	N_1	N_2	N_2	N_2	N_3	N_1	N_1
BFD	N_0	N_0	N_0	N_3	N_3	N_4	N_1	N_2	N_2	N_2	N_3	N_1	N_1
FFDACO	N_2	N_2	N_2	N_3	N_1	N_1	N_4	N_4	N_4	N_4	N_1	N_3	N_0
BFDACO	N_0	N_0	N_0	N_3	N_1	N_1	N_4	N_3	N_4	N_4	N_2	N_3	N_1

average value in a period of time, which is different from that of the simulation experiment. Thus, the real cluster and simulation experiments have different results. However, the variation patterns of the fitness function values of different algorithms are the same.

4.3 Fault injection experiment and analysis

The fault injection technology[37–39] is used to evaluate the system failure rate when the cloud system adopts the VM fault-tolerant placement algorithm.

On the basis of the assumption of system failure mode in Section 2.1 and the VM placement example in Section 4.2, faults are injected into the SQL and data nodes. The injected faults include permanent and transient faults. The permanent faults are injected at the software or VM level. The software fault is simulated by killing its daemon, and the VM fault is simulated by shutting down the VM. The transient fault is simulated by modifying the return value of the system call, such as ioctl().

According to Eq. (5), the function of the system failure rate is expressed as Eq. (21):

$$f = \frac{\text{Fail}_{\text{num}}}{\text{Fail}_{\text{num}} + \text{Pass}_{\text{num}}} \tag{21}$$

where Fail_{num} represents the number of failed requests for RUBiS service and Pass_{num} represents the number of passed requests for RUBiS service.

We access the RUBiS service 100 000 times in experiments. Faults are injected when the number of times that the service is accessed reaches half. Table 2 shows the failure rate of the prototype system injected with permanent faults. NFFDACO or NBFDACO represents the nonredundent VM placement algorithm based on FFDACO or BFDACO.

Notably, if critical VMs place their redundant replicas on the basis of the proposed method, then the system's failure rate will be considerably reduced. When a critical VM fails, its redundant replica will take over and continue to provide service in a short period of time. Thus, the number of failed requests for the RUBiS service can be considerably reduced.

Table 3 shows the failure rate of the prototype system injected with transient faults in critical VMs. Notably, the system that adopts the proposed method has a low failure rate. When a critical VM cannot handle the request because of transient failure, its redundant replica will take over and continue to provide service. The number of failed requests for the RUBiS service is reduced.

Table 2 Failure rate of the prototype system injected with permanent faults.

Method	Failure rate	
	MySQL	KVM
NFFDACO	0.5149	0.4937
FFDACO	0.0450	0.0390
NBFDACO	0.4858	0.5041
BFDACO	0.0520	0.0570

Table 3 Failure rate of the prototype system injected with transient faults.

Method	Failure rate
NFFDACO	0.014
FFDACO	0.005
NBFDACO	0.015
BFDACO	0.003

If permanent faults are injected into the database or its corresponding VM, then the database cannot provide services and the access request for the RUBiS service will fail. Thus, the failure rate shown in Table 2 for the system based on the NFFDACO or NBFDACO is approximately 0.5. When a transient fault is injected into a VM, its failure rate and the number of failed requests for the RUBiS service are reduced. Therefore, the failure rate shown in Table 3 is lower than that shown in Table 2.

The simulation, real cluster, and fault injection experiments show that the VM fault-tolerant placement solution obtained by the method proposed in this study can not only achieve the optimal solution under multiple VM fault-tolerant placement factors but also have a low system failure rate.

5 Conclusion

The VM resource management of cloud computing systems in data centers is a hot spot of current research. VM static placement and VM dynamic management are two types of VM management methods in data centers. For the VM static placement problem, most of the existing research only considers how to map VMs to physical nodes under certain constraints and rarely addresses the reliability problem. For VM static placement in star topological data centers, this study proposes a multi-objective optimization method of initial VM fault-tolerant placement. In this work, on the basis of the structural characteristics of the cloud computing system, a VM fault-tolerant management system is established at the unified resource layer to control the VM fault-tolerant placement process; on the basis of multiple factors, a multi-objective optimization model of initial VM fault-tolerant placement is proposed; and

a heuristic ant colony algorithm is proposed to solve the multi-objective optimization model. The simulation, real cluster, and fault injection experiments show that the proposed method can obtain better VM fault-tolerant placement solution than the traditional methods. However, the VM fault-tolerant placement method proposed in this study only considers the case of single VM failure. Moreover, in this study, we only construct a model to solve the initial VM fault-tolerant placement problem. How to combine initial VM fault-tolerant placement with dynamic VM fault-tolerant management to perform full life cycle VM management and how to deal with multiple VM failures will be the focus of our future works.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (Nos. 61432017 and 61772199).

References

[1] I. Foster, Y. Zhao, I. Raicu, and S. Y. Lu, Cloud computing and grid computing 360-degree compared, in *Proc. 2008 Grid Computing Environments Workshop*, Austin, TX, USA, 2008, pp. 1–10.

[2] H. L. Chen, A qualitative and quantitative study on availability of cloud computing, http://www.valleytalk.org/wp-content/uploads/2013/10/, 2013.

[3] M. Nelson, B. H. Lim, and G. Hutchins, Fast transparent migration for virtual machines, in *Proc. 2005 USENIX Annu. Technical Conf.*, Anaheim, CA, USA, 2005, pp. 391–394.

[4] M. Lee, A. Krishnakumar, P. Krishnan, N. Singh, and S. Yajnik, Hypervisor-assisted application checkpointing in virtualized environments, in *Proc. IEEE/IFIP 41st Int. Conf. Dependable Systems & Networks*, Hong Kong, China, 2011, pp. 371–382.

[5] X. Zhang, Z. G. Huo, J. Ma, and D. Meng, Fast and live whole-system migration of virtual machines, (in Chinese), *Journal of Computer Research and Development*, vol. 49, no. 3, pp. 661–668, 2012.

[6] F. Xu, F. M. Liu, L. H. Liu, H. Jin, B. Li, and B. C. Li, iAware: Making live migration of virtual machines interference-aware in the cloud, *IEEE Transactions on Computers*, vol. 63, no.12, pp. 3012–3025, 2014.

[7] K. J. Ye, Z. H. Wu, C. Wang, B. B. Zhou, W. S. Si, X. H. Jiang, and A. Y. Zomaya, Profiling-based workload consolidation and migration in virtualized data centers, *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 3, pp. 878–890, 2015.

[8] H. K. Liu and B. S. He, VMbuddies: Coordinating live migration of multi-tier applications in cloud environments, *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 4, pp. 1192–1205, 2015.

[9] J. Zhu, W. Dong, Z. F. Jiang, X. G. Shi, Z. Xiao, and X. M. Li, Improving the performance of hypervisor-based fault tolerance, in *Proc. IEEE Int. Symp. Parallel & Distributed Processing*, Atlanta, GA, USA, 2010, pp. 1–10.

[10] J. Zhu, Z. F. Jiang, Z. Xiao, and X. M. Li, Optimizing the performance of virtual machine synchronization for fault tolerance, *IEEE Transactions on Computers*, vol. 60, no. 12, pp. 1718–1729, 2011.

[11] D. Shen, J. Z. Luo, F. Dong, and J. X. Zhang, VirtCo: Joint coflow scheduling and virtual machine placement in cloud data centers, *Tsinghua Science and Technology*, vol. 24, no. 5, pp. 630–644, 2019.

[12] Z. Liu, S. J. Sun, J. Xing, Z. Fu, X. H. Hu, J. W. Pi, X. F. Yang, Y. S. Lu, and J. Li, MN-SLA: A modular networking SLA framework for cloud management system, *Tsinghua Science and Technology*, vol. 23, no. 6, pp. 635–644, 2018.

[13] 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, (in Chinese), *Chinese Journal of Computers*, vol. 34, no. 12, pp. 2253–2264, 2011.

[14] K. Tsakalozos, M. Roussopoulos, and A. Delis, Hint-based execution of workloads in clouds with Nefeli, *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 7, pp. 1331–1340, 2013.

[15] F. Farahnakian, A. Ashraf, T. Pahikkala, P. Liljeberg, J. Plosila, I. Porres, and H. Tenhunen, Using ant colony system to consolidate VMs for green cloud computing, *IEEE Transactions on Services Computing*, vol. 8, no. 2, pp. 187–198, 2015.

[16] E. G. Coffman, M. R. Garey, and D. S. Johnson, Approximation algorithms for bin packing: A survey, in *Approximation Algorithms for NP-Hard Problems*. Boston, MA, USA: PWS Publishing, 1997, pp. 46–93.

[17] N. Bobroff, A. Kochut, and K. Beaty, Dynamic placement of virtual machines for managing SLA violations, in *Proc. 10th IFIP/IEEE Int. Symp. Integrated Management*, Munich, Germany, 2007, pp. 119–128.

[18] S. Chaisiri, B. S. Lee, and D. Niyato, Optimization of resource provisioning cost in cloud computing, *IEEE Transactions on Services Computing*, vol. 5, no. 2, pp. 164–177, 2012.

[19] C. H. Lien, Y. W. Bai, and M. B. Lin, Estimation by software for the power consumption of streaming-media servers, *IEEE Transactions on Instrumentation and Measurement*, vol. 56, no. 5, pp. 1859–1870, 2007.

[20] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari, Server workload analysis for power minimization using consolidation, in *Proc. 2009 Conf. USENIX Annu. Technical Conf.*, San Diego, CA, USA, 2009, p. 28.

[21] J. K. Dong, H. B. Wang, and S. D. Cheng, Energy-performance tradeoffs in IaaS cloud with virtual machine scheduling, *China Communications*, vol. 12, no. 2, pp. 155–166, 2015.

[22] X. Jing and J. A. B. Fortes, Multi-objective virtual machine placement in virtualized data center environments, in *Proc. 2010 IEEE/ACM Int'l Conf. Green Computing and Communications & Int'l Conf. Cyber, Physical and Social Computing*, Hangzhou, China, 2010, pp. 179–188.

[23] F. Ma, F. Liu, and Z. Liu, Multi-objective optimization

for initial virtual machine placement in cloud data center, *Journal of Information and Computational Science*, vol. 9, no. 16, pp. 5029–5038, 2012.

[24] S. N. Wang, H. X. Gu, and G. Wu, A new approach to multi-objective virtual machine placement in virtualized data center, in *Proc. IEEE 8th Int. Conf. Networking, Architecture and Storage*, Xi'an, China, 2013, pp. 331–335.

[25] F. Machida, M. Kawato, and Y. Maeno, Redundant virtual machine placement for fault-tolerant consolidated server clusters, in *Proc. 2010 IEEE Network Operations and Management Symposium*, Osaka, Japan, 2010, pp. 32–39.

[26] G. Y. Jung, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu, Performance and availability aware regeneration for cloud based multitier applications, in *Proc. IEEE/IFIP Int. Conf. Dependable Systems and Networks*, Chicago, IL, USA, 2010, pp. 497–506.

[27] E. Bin, O. Biran, O. Boni, E. Hadad, E. K. Kolodner, Y. Moatti, and D. H. Lorenz, Guaranteeing high availability goals for virtual machine placement, in *Proc. 31st Int. Conf. Distributed Computing Systems*, Minneapolis, MN, USA, 2011, pp. 700–709.

[28] D. Epping and F. Denneman, *VMware vSphere 4.1 HA and DRS Technical Deepdive*. North Charleston, SC, USA: CreateSpace, 2010, pp. 15–22.

[29] Z. B. Zhen, T. C. Zhou, M. R. Lyu, and I. King, FTCloud: A component ranking framework for fault-tolerant cloud applications, in *Proc. IEEE 21st Int. Symp. Software Reliability Engineering*, San Jose, CA, USA, 2010, pp. 398–407.

[30] Z. B. Zheng, T. C. Zhou, M. R. Lyu, and I. King, Component ranking for fault-tolerant cloud applications, *IEEE Transactions on Services Computing*, vol. 5, no. 4, pp. 540–550, 2012.

[31] F. Hermenier, J. Lawall, and G. Muller, BtrPlace: A flexible consolidation manager for highly available applications, *IEEE Transactions on Dependable and Secure Computing*, vol. 10, no. 5, pp. 273–286, 2013.

[32] X. X. Cui, *Multi-Objective Evolutionary Algorithms and Their Applications*, (in Chinese). Beijing, China: National Defense Industry Press, 2006, pp. 10–20.

[33] T. Stützle and H. H. Hoos, MAX-MIN ant system, *Future Generation Computer Systems*, vol. 16, no. 8, pp. 889–914, 2000.

[34] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, *Software*: *Practice and Experience*, vol. 41, no.1, pp. 23–50, 2011.

[35] E. Cecchet, A. Chanda, S. Elnikety, J. Marguerite, and W. Zwaenepoel, Performance comparison of middleware architectures for generating dynamic web content, in *Proc. 2003 ACM/IFIP/USENIX Int. Conf. Middleware*, Rio de Janeiro, Brazil, 2003, pp. 242–261.

[36] Y. Tamura, Kemari: Fault tolerant VM synchronization based on KVM, https://www.linux-kvm.org/images/0/0d/0.5.kemari-kvm-forum-2010.pdf, 2010.

[37] A. Jin and J. H. Jiang, Fault injection scheme for embedded systems at machine code level and verification, in *Proc. 15th IEEE Pacific Rim Int. Symp. Dependable Computing*, Shanghai, China, 2009, pp. 55–62.

[38] J. W. Hu and J. H. Jiang, Design and implementation of a fault injection mechanism for software reliability evaluation, (in Chinese), *Journal of Computer-Aided Design & Computer Graphics*, vol. 24, no. 6, pp. 741–751, 2012.

[39] D. Q. Zhang, J. H. Jiang, and L. B. Chen, A method for validating the effectiveness of fault clustering and failure clustering of programs, *Scientia Sinica Informationis*, vol. 44, no. 10, pp. 1323–1344, 2014.

Jianhui Jiang received the BE, ME, and PhD degrees in traffic information engineering and control from Shanghai Tiedao University (in April 2000, it was merged to Tongji University) in 1985, 1988, and 1999, respectively. During 1994–2000, he was an associate professor in computer science and technology at Shanghai Tiedao University. Since 2000, he has been a full professor in computer science and technology at Tongji University. During 2007–2011, he was the chair of the Department of Computer Science and Technology, Tongji University. Since 2011, he is the associate dean of the School of Software Engineering, Tongji University. He is the vice director of Technical Committee on Fault-tolerant Computing, Chinese Computer Federation (CCF). He has served on several program committees of national or international symposiums or workshops. He has co-authored two books and published more than 200 technical papers. His current research interests include dependable systems and networks, software reliability engineering, and VLSI/SoC testing and fault-tolerance. He is a senior member of CCF.

Wei Zhang received the BS degree in software engineering from Shanghai Institute of Technology and ME degree in software engineering from Tongji University, Shanghai, China in 2015 and 2018, respectively. Now he is a PhD candidate at Tongji University. His current research interests include dependable systems and networks and software reliability engineering.

Xiao Chen received the BS degree in information and computing science, ME degree in computer science and technology from China Three Gorges University, and PhD degree in software engineering from Tongji University, China in 2009, 2012, and 2016, respectively. His current research interests include software reliability engineering and fault-tolerant computing.