Future Internet **2015**, *7*, 465-483; doi:10.3390/fi7040465

OPEN ACCESS

future internet

ISSN 1999-5903

www.mdpi.com/journal/futureinternet

Article

Dynamic Load Balancing Strategy for Cloud Computing with Ant Colony Optimization

Ren Gao [1,*] and Juebo Wu [2,3]

[1] School of Information Engineering, Hubei University of Economics, Wuhan 430205, China

[2] Department of Geography, National University of Singapore Arts Link, Singapore 117570, Singapore; E-Mail: wujuebo@gmail.com

[3] ZTE ICT Technologies Co. Ltd., ZTE Corporation, Shenzhen 518057, China

* Author to whom correspondence should be addressed; E-Mail: gaoren@whu.edu.cn; Tel.: +86-137-9701-9320.

Academic Editor: Xiaolong Li

Received: 22 September 2015 / Accepted: 11 November 2015 /Published: 26 November 2015

Abstract: How to distribute and coordinate tasks in cloud computing is a challenging issue, in order to get optimal resource utilization and avoid overload. In this paper, we present a novel approach on load balancing via ant colony optimization (ACO), for balancing the workload in a cloud computing platform dynamically. Two strategies, forward-backward ant mechanism and max-min rules, are introduced to quickly find out the candidate nodes for load balancing. We formulate pheromone initialization and pheromone update according to physical resources under the cloud computing environment, including pheromone evaporation, incentive, and punishment rules, *etc.* Combined with task execution prediction, we define the moving probability of ants in two ways, that is, whether the forward ant meets the backward ant, or not, in the neighbor node, with the aim of accelerating searching processes. Simulations illustrate that the proposed strategy can not only provide dynamic load balancing for cloud computing with less searching time, but can also get high network performance under medium and heavily loaded contexts.

Keywords: load balancing; cloud computing; ant colony optimization; swarm intelligence

1. Introduction

Cloud computing is increasingly being adopted by large businesses, as well as small and medium sized businesses, for "on-demand" and "utility computing", which holds huge promise for the future of service computing [1]. Virtualization is a key enabling technology for cloud computing environments, which makes it possible to run multiple operating systems and multiple applications on the same hardware at the same time, so as to provide services by a virtual unit [2]. Through virtualization technology, not only can overall hardware utilization improve and lower costs for disaster recovery, but it can also achieve automatic monitoring for all hosts. However, it is very difficult to assign a large number of tasks to dynamic resources for distributed computing. There are a variety of factors that may lead to some nodes in the overload state while others remain in the underload state, such as uneven allocation of resources, user needs changing over time, newly joining nodes, and a high likelihood of failure in the overload nodes, *etc.* [3–5].

Load balancing is the most effective way to solve the above problem in a cloud computing infrastructure, which ensures that services are delivered transparently regardless of the physical implementation and location within the "cloud". In recent decades, great progress has been achieved for load balancing, and one of the most promising branches is swarm intelligence algorithms, such as ant colony optimization [6–8], artificial bee colony [9,10], particle swarm optimization [11,12], *etc.* Ant colony optimization, proposed by Marco Dorigo in 1992 [13], is a class of stochastic optimization algorithms based on the actions of an ant colony. By analyzing the previous work of ACO, we found that the ant colony optimization is suitable for load balancing applications in cloud computing because [14–16]: (1) the ant colony is able to crawl among different nodes to search for the optimal solution in cloud computing infrastructure; (2) the ACO is a kind of parallel mechanism that can be applied in distributed computing with high performance; and (3) it is a self-organizing algorithm based on the local information to make judgments and actions, that is, the system does not require a global control center.

The motivation of this paper is to establish a load balancing mechanism which utilizes ACO to balance the tasks among nodes in cloud computing. The targets, including overload or underload nodes, will be quickly identified to operate load balancing by two types of ants with their communications by pheromones. The rest of the paper is structured as follows. Section 2 outlines the related work for load balancing algorithms. The dynamic load balancing strategy is presented for cloud computing in Section 3. Section 4 describes the details of the proposed strategy using the improved ACO. Simulation and results analysis are carried out in Section 5. At the end, a brief summary is given along with the future work.

2. Related Work

Load balancing plays an essential role in providing quality of service (QoS) guarantees in cloud computing, and it has been generating substantial interest in the research community. There are a great deal of approaches that have coped with the load balancing problem in cloud computing. We discuss the previous related work of load balancing by dividing them into two classes according to the underlying algorithm.

The first class consists of diverse conventional approaches without utilizing any kind of swarm intelligence algorithms.

Many load balancing approaches were proposed in recent years and each focused on different aspects of algorithms and policies, e.g., leveraging a central load balancing policy for virtual machines [17], the scheduling strategy on load balancing of virtual machine (VM) resources based on genetic algorithms [18], a mapping policy based on multi-resource load balancing for virtual machines [19], adaptive distributed algorithm for virtual machines [20], weighted least-connection strategy [21], and two-phase scheduling algorithms [22]. Additionally, several methods of load balancing were presented for different cloud applications, for example, a service-based model for large scale storage [23], data center management architecture [24], and a heterogeneous cloud [25]. Although these contributions have made great progress in load balancing under cloud computing, it has a high degree of centralization and is not easy to extend. Furthermore, these presented approaches did not fully reflect the characteristics of resource nodes and are more suitable to the static situation of cloud computing.

The second class contains approaches use swarm intelligence algorithms, such as ant colony optimization [6–8], artificial bee colony [9,10], and particle swarm optimization [11,12], which is better for the dynamic situation of cloud computing.

With self-organized behavior, these social insects can be imitated as such, or with necessary modifications, to solve analogous problems in cloud computing. In [6], Nishant, K. *et al.* proposed an algorithm for load distribution of a workload with a modified approach of ACO from the perspective of cloud or grid network systems. In this approach, the ants only updated a single result set continuously in the process, rather than updating their own result set. In [7], a load balancing mechanism was proposed based on ant colony and complex network theory in an open cloud computing federation. This is the first time that ACO and complex networks were introduced together into load balancing in cloud computing and obtained good performance. In [8], Mishra, R. *et al.* gave a solution to load balancing in the cloud by ACO, to maximize or minimize different performance parameters, such as CPU load and memory capacity. However, few factors were considered as pheromones to find target nodes when using ACO in the above three approaches.

In [9,10], Sesum-Cavic, V. *et al.* presented a novel approach for load balancing based on artificial bee colony. A generic architecture, named SILBA (self-initiative load balancing agents), was defined to support the exchange of different algorithms through plugging techniques. Six algorithms were applied in this architecture and the results demonstrated promising benefits in the Amazon EC2 cloud. Although SILBA is a good pattern, it did not take into account the lower demands for node servers in cloud computing environments and the dynamic user needs.

Particle swarm optimization (PSO) was also adopted for load balancing in cloud computing, such as [11,12]. In [11], it proposed a new task scheduling model to avoid the serious load imbalance problem, with improvement of the standard PSO by introducing a simple mutation mechanism and a self-adapting inertia weight method. To solve the optimization problem of discrete space in cloud computing, Feng, X. *et al.* constructed an appropriate resource-task model based on a discrete particle swarm optimization algorithm [12]. The experiment results showed that the discussed PSO methods can enhance the utilization in load balancing of resources ,but they may take a large amount of time with a huge number of tasks.

Other applications and research on load balancing with swarm intelligence algorithms can be found in [26–33]. Since these algorithms were originally designed for distributed load balancing rather than cloud computing, much work needs to be done if we want to apply these algorithms into cloud computing.

In this paper, we focus on how to utilize ACO to establish a model of dynamic load balancing for cloud computing, while fully considering the characteristics of cloud computing itself. Moreover, we emphasize on well-defined strategies of pheromone update, in order to not only avoid falling into a local optimum, but also improve the convergence speed and accuracy.

3. Dynamic Load Balancing Strategy in Cloud Computing

Master-slave architecture is a mature architecture with a single master server or job tracker and several slave servers, which has been widely used in cloud computing like Google's MapReduce and Hadoop. Figure 1 shows the typical scenario of network topology of virtual resources in cloud computing, which is based on the master-slave architecture and the cloud platform discussed in this paper.

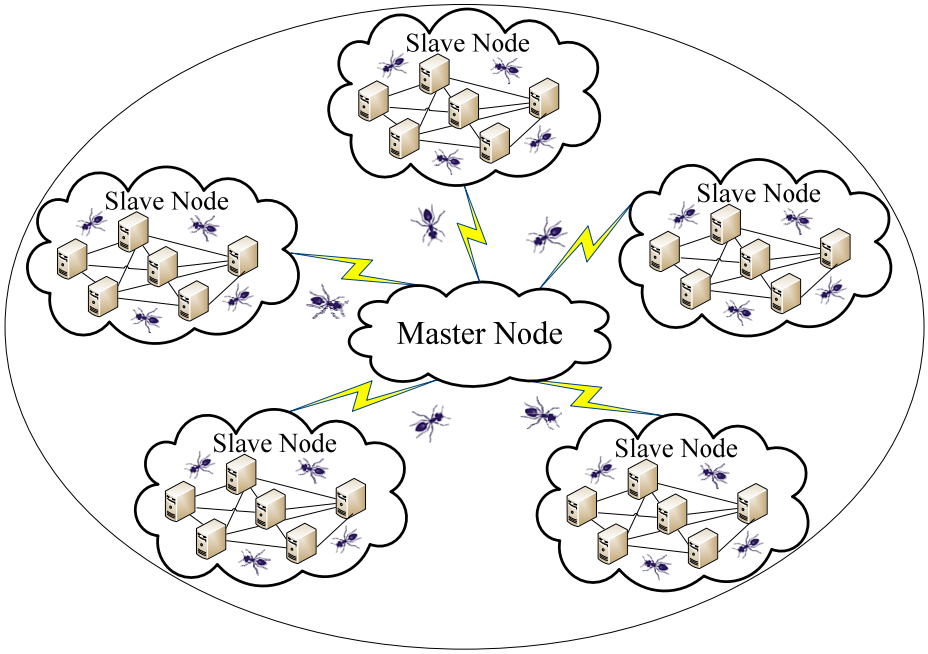

Figure 1. Network topology of virtual resources in cloud computing.

In master-slave architecture, a job is firstly submitted to a master node by the user. Then the job is divided into several executable tasks in the master node and the generated tasks are distributed to different slave nodes. After that, the tasks are executed in the slave nodes separately with the guidance of the master node, and the results are returned to the master node. Finally, the distributed results are combined together in the master node and sent to the requesting user. Furthermore, the master node is responsible for monitoring the whole steps and re-executing the failed tasks.

During this process, the uneven distribution of tasks may cause that some slave nodes are in light load conditions while others are in heavy load conditions. In this case, load balancing operation ought to be carried out dynamically for the cloud platform in order to keep the platform stable and operating efficiently. By analyzing the cloud computing platform, there are several characteristics in common with ACO, such as cloud nodes being analogous to food locations, data repositories to nests, and load allocation to foraging activity.

In our approach, the main procedure of load balancing with ACO consists of two steps before load balancing execution as below.

(1) Ant generation. Check the cloud platform periodically, and generate ants if and only if there are existing overload nodes or underload nodes; and

(2) To find target nodes. According to searching rules, the ant is looking for the target nodes which meet the conditions of load balancing in its surrounding area. The target node is also called the candidate node for load balancing.

For fast convergence, we employ two diverse ants in search of the slave nodes. At the same time, we leverage max-min rules to trigger ant generation, so as to improve the efficiency of the algorithm.

3.1. Forward-Backward ant Mechanism

The ants are divided into two categories: forward ant and backward ant, which is the same mechanism described in [34] but with distinct definition. The forward ant is responsible to find the candidate nodes for load balancing in cloud computing platform and it starts the searching activity from its generated node. The candidate nodes include overload nodes and underload nodes. The backward ant is in charge of updating information pheromones for the path as that of its corresponding forward ant, but in the opposite direction. The backward ant is generated at each time when the forward ant identifies a candidate node.

As described in [34], the forward ant calculates the moving probability for each neighbor before it moves and then chooses the largest one as its next destination. To accelerate convergence, we add a special strategy to our model when the forward ant meets the backward ant in the same node, that is, the moving probability is computed by considering both the information pheromone of the node itself and the information pheromone from all backward ants in this node.

To simulate the meeting process, we make use of a timer to record the life cycle of a backward ant after it is produced. Some storage units are set for each node and they are used to save the information pheromone carried by backward ants, with one unit for one backward ant. Two types of ants are regarded as meeting each other only when the forward ant arriving at one node before the timer of one backward ant running out in this node. The information pheromone carried by a backward ant will be cleared when the timer reaching zero. When there is more than one backward ant in one node, all the influences by these backward ants should be considered when computing the moving probability of forward ant in this node.

The details about meeting handling and moving probability will be discussed in Section 4.

3.2. Max-Min Rules

We define two distinct rules, named max-min rules, to trigger the forward ant generation, with the purpose of reducing the time for searching candidate nodes as below.

Rule 1: Maximum value trigger rule. A forward ant is generated from a slave node when the load in this node is greater than a certain threshold. It indicates that the node has been running close to or beyond its maximum load, which needs to distribute the tasks to idle nodes so as to achieve optimal resource utilization.

Rule 2: Minimum value trigger rule. A forward ant is generated from a slave node when the load in this node is smaller than a certain threshold. It denotes that the node is running in the light load state, which can accept a range of new tasks, in order to share its resource to the overload nodes.

3.3. Process of Load Balancing

Based on the above strategies, the core steps of load balancing are described as follows.

(1) Compute moving probability for all of its neighbors and select the biggest one as its next destination;
(2) Move to a new node and judge whether it is a candidate node. If yes, generate a backward ant and initialize this backward ant. For forward ant, go to step 1;
(3) The backward ant goes back to the starting point of its forward ant, along the path of its forward ant with the opposite direction. Update the information pheromone of each node the backward ant passes by, and delete the backward ant when reaching the starting point;
(4) Calculate the sum resources of the candidate nodes, and stop the process if they are able to meet the demand of load balancing and
(5) Perform the load balancing operation.

These steps are the same for max-min rules except the way to calculate the moving probability. The details will be discussed in Section 4.

4. Dynamic Load Balancing Modeling with ACO

With respect to the strategy presented above, we further analyze the details of the moving probability with ant colony optimization in this section. To explore both load allocation efficiency and network performance, two critical issues must be addressed. First, pheromone initialization should be reasonable in the cloud computing environment, to satisfy the desired QoS. Second, the pheromone update has to meet the dynamic demand of the workload variability, with the aim of accelerating the convergence.

4.1. Pheromone Initialization

In cloud computing, the physical resources allocated to each virtual node are not the same and usually changing dynamically. Due to of this characteristic, we use the physical resources of virtual machines to measure a node's initial pheromone, as described in [15,16]. Five physical resources are involved in pheromone initialization here, that is, CPU (number of cores and MIPS for each core), internal storage, external storage, I/O interface, and bandwidth. The CPU capability can be calculated by:

$$p_{CPU} = n \times p \tag{1}$$

To facilitate the calculation, we set an upper limit for each parameter. When a parameter exceeds its upper limit, the limit value is chosen instead of the actual value. The limits are given in Equation (2).

$$p_{CPU} \le p_{\max}, m_i \le m_{i\max}, m_e \le m_{e\max}, p_{i/o} \le p_{i/o\max}, p_b \le p_{b\max} \tag{2}$$

The capability definitions of physical resources of virtual machines are defined as blow.
For CPU:

$$\tau_{CPU}(0) = \frac{p_{CPU}}{p_{\max}} \times 100\% \tag{3}$$

For internal storage:

$$\tau_{mi}(0) = \frac{m_i}{m_{i\max}} \times 100\% \tag{4}$$

For external storage:

$$\tau_{me}(0) = \frac{m_e}{m_{e\max}} \times 100\% \tag{5}$$

For I/O interface:

$$\tau_{i/o}(0) = \frac{p_{i/o}}{p_{i/o\max}} \times 100\% \tag{6}$$

For bandwidth:

$$\tau_b(0) = \frac{p_b}{p_{b\max}} \times 100\% \tag{7}$$

Therefore, we define the pheromone initialization for one slave node as:

$$\begin{aligned} &\tau_i = \psi_1 \tau_{CPU}(0) + \psi_2 \tau_{mi}(0) + \psi_3 \tau_{me}(0) + \psi_4 \tau_{i/o}(0) + \psi_5 \tau_b(0) \\ &\sum_{n=1}^{5} \psi_n = 1 \end{aligned} \tag{8}$$

where ψ_n is a weight coefficient, which is used to adjust the influence of the physical resources in cloud computing.

4.2. Pheromone Update

The goal of pheromone update in our approach is to increase the pheromone values for slave nodes associated with good conditions and decrease those associated with bad ones. Three factors that impact the pheromone update are considered in our strategy, namely pheromone evaporation, update by task, and incentives for successful tasks.

4.2.1. Pheromone Evaporation

According to [13], the pheromone in the node is decreasing over time due to evaporation. We use the local update strategy to modify the pheromone on slave nodes where the pheromone is not zero. The pheromone update by evaporation is defined as:

$$\tau_i(t+1) = (1-\rho) \times \tau_i(t), \ 0 < \rho < 1 \tag{9}$$

where $\tau_i(t)$ is the pheromone in slave node *i* at *t* moment and ρ is a coefficient of evaporation.

4.2.2. Pheromone Evaporation by New Task

The capability of a slave node is changed when a new task is allocated to this node. After receiving new tasks, the capability of this slave node decreases because of the resources being consumed. In this case, the pheromone update is obtained by:

$$\tau_i(t+1) = (1-\mu) \times \tau_i(t), \ 0 < \mu < 1 \tag{10}$$

where μ is the factor to adjust the degree of resources consuming.

The capability of a slave node is also changing when a new task is performing as time goes by. The capability of this slave node increases owing to the releasing of resources. In this case, the pheromone is increased by:

$$\tau_i(t+1) = (1+\nu) \times \tau_i(t), \ 0 < \nu < 1 \tag{11}$$

where ν is a factor to control how fast the resources are released.

4.2.3. Pheromone Evaporation for Backward Ant

As mentioned in Section 3.1, the pheromones of backward ants are stored in slave nodes, which are controlled by the relevant timers. Like normal pheromones of each node, the pheromones of backward ants are evaporated over time. The pheromone update by evaporation for backward ants is defined as:

$$\tau_i(t+1) = (1+\pi) \times \tau_i(t), \ 0 < \pi < 1 \tag{12}$$

where π is a coefficient of evaporation for backward ant.

4.2.4. Incentive and Punishment Rules

We define two rules for task execution in a slave, that is, incentive rule and punishment rule. The former means that the pheromone is increased in this node if the tasks are performed successfully. The latter denotes that the pheromone is decreased in this node if the tasks are done unsuccessfully. The pheromone update by these two rules can be calculated by:

$$\begin{aligned} &\tau_i(t+1) = (1+\theta) \times \tau_i(t), \\ &if\,(success) \quad 0 < \theta < 1 \\ &else \quad -1 < \theta < 0 \end{aligned} \tag{13}$$

where θ is a factor to adjust the degree of incentive and punishment for slave nodes.

4.3. Task Execution Prediction

Task execution prediction is to evaluate the execution velocity for a slave node, which reflects the capability and performance of virtual resources in cloud computing. Generally speaking, the overall efficiency of the cloud platform can be improved when allocating the tasks to the slave nodes with better performance. We design a prediction model that evaluates the execution velocity of a slave node for the next time frame by accumulating the previous records. Through the current workload and the workload performed last time, we can predict the velocity of a slave node for the next time frame by:

$$EV_i^{a_{k+1}}(k+1) = \frac{a_{k+1}}{a_k}((1-\omega)EV_i^{a_k}(k) + \omega RV_i^{a_k}(k)) \tag{14}$$

where $EV_i^{a_k}(k)$ is the *k*th velocity of task execution prediction in slave node *i* (MIPS), and a_k is the *k*th workload in this node. $RV_i^{a_k}(k)$ is the *k*th real execution velocity. ω is an adjustable parameter, which is used to control the weight of real execution and prediction.

As time goes by, the current tasks become less and less while the predicting task execution becomes much faster. Therefore, the velocity of task execution prediction should be increased during tasks running at regular intervals, defined as:

$$EV_i^{a_{k+1}}(t+1) = (1+\sigma)EV_i^{a_{k+1}}(t) \tag{15}$$

where θ is an adjust table parameter to control the increasing degree by task execution prediction.

4.4. Moving Rules for Forward Ant

In our method, there are two cases when the forward ant computes the moving probability for the next destination. One is that no backward ant is appearing in neighbor nodes. The other is that one or more than one backward ant is appearing in neighbor nodes. We explore two such cases, respectively, in this section.

4.4.1. No Backward Ant Appearing in Neighbor Nodes

As described in Section 3.2, the generation of forward ant is triggered in diverse conditions for searching candidate nodes. Thus, we discuss the case with no backward ant appearing in the neighbor nodes from two aspects.

(1) The forward ant triggered by overload node

In this case, the moving probability of forward ant is calculated by:

$$p_{ij} = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha}[EV_{ij}^{a_{k+1}}(t)]^{\beta}[\eta_{ij}(t)]^{\gamma}[D_{ij}(t)]^{\kappa}}{\sum_{j\in N_n}\{[\tau_{ij}(t)]^{\alpha}[EV_{ij}^{a_{k+1}}(t)]^{\beta}[\eta_{ij}(t)]^{\gamma}[D_{ij}(t)]^{\kappa}\}} & if \quad j \in N_n \\ 0 \end{cases} \tag{16}$$

where p_{ij} is the moving probability of the forward ant from node *i* to node *j*. N_n is the set of unvisited neighbor nodes. $\tau_{ij}(t)$ is the pheromone of node *j*. $EV_{ij}^{a_{k+1}}(t)$ is the velocity of task execution prediction in slave node *j*. $\eta_{ij}(t)$ is the cost from node *i* to node *j*, that is, $1/d_{ij}$. $D_{ij}(t)$ is the proportion of the degree of node *j* to the sum degree of all neighbor nodes, defined as:

$$D_{ij} = \frac{d_j}{\sum_{k\in NB_i} d_k} \tag{17}$$

where NB_i is the neighbor set of node *i*. This means that the node with the maximum degree ought to be more likely to be chosen as the next destination. α, β, γ, and κ stand for the weights of different parameters.

(2) The forward ant triggered by underload node

In this context, the moving probability of forward ant is calculated by:

$$p_{ij} = \begin{cases} \dfrac{[\eta_{ij}(t)]^{\gamma}[D_{ij}(t)]^{\kappa}}{\sum_{j\in N_n}[\eta_{ij}(t)]^{\gamma}[D_{ij}(t)]^{\kappa}} & if \quad j \in N_n \\ 0 \end{cases} \tag{18}$$

where the parameters have the same meanings as in Equation (16) except for removing $\tau_{ij}(t)$ and $EV_{ij}^{a_{k+1}}(t)$. For the forward ant triggered by underload node, the ant's goal is not to find the nodes with idle resources in the network, but to find the overload nodes. This means that the ant selects a next destination only by the cost in the edge between node *i* and node *j*.

4.4.2. Meeting Backward Ant in Neighbor Node

As described in Section 3.1, the pheromone carried by the backward ant should be considered when the forward ant meeting the backward ant in the same node. Since there may be more than one ant in one node, the total pheromone in this node has to contain the pheromone from all backward ants. Like Section 4.4.1, two situations should be discussed separately.

(1) The forward ant triggered by overload node

The moving probability is defined as:

$$p_{ij} = \begin{cases} \dfrac{[\tau_{ij}(t) + \sum \tau_b(t)]^{\alpha}[EV_{ij}^{a_{k+1}}(t)]^{\beta}[\eta_{ij}(t)]^{\gamma}[D_{ij}(t)]^{\kappa}}{\sum_{j \in N_n}[\tau_{ij}(t) + \sum \tau_b(t)]^{\alpha}[EV_{ij}^{a_{k+1}}(t)]^{\beta}[\eta_{ij}(t)]^{\gamma}[D_{ij}(t)]^{\kappa}} & if\ j \in N_n \\ 0 \end{cases} \quad (19)$$

where $\sum \tau_b(t)$ is the sum of pheromones carried by the backward ant and the other parameters are the same meanings as Equation (16).

(2) The forward ant triggered by underload node

As the method described in Section 4.4.1, the moving probability is computed by Equation (18).

4.5. Dynamic Load Balancing Algorithm

Algorithm 1 presents the essential steps to achieve load balancing by the use of the proposed approach. At the beginning, all slave nodes in cloud computing platform are initialized using Equation (7) (line 2 of Algorithm 1), the master node gets the job from the user's request, and the job is divided into a number of executable tasks (lines 3 and 4 of Algorithm 1). Then, the tasks are distributed to slave nodes for execution by n_t times, since the slave nodes cannot execute all the tasks at once (line 6 of Algorithm 1). After that, the nodes are judged if there is any overload or underload node (line 7 of Algorithm 1). If yes, the forward ant is created and carries out the searching missions, along with the assistance of the backward ant (lines from 7 to 21 of Algorithm 1). In this process, we should note that the moving probability is calculated according to diverse situations, that is, whether the forward ant meets the backward ant or not. If the candidate nodes can satisfy the demand of performing load balancing, do it and continue the next loop (lines from 22 to 25 of Algorithm 1). If the tasks have been done, go on to the next group of new tasks (lines 26 and 27 of Algorithm 1).

Algorithm 1 The step by step procedure of the proposed load balancing

1. Beginning of proposed algorithm
2. Initialize pheromone for slave nodes;
3. Get-job-from-user (job_n) for master node;
4. Job-divides-into-tasks (job_n) by master node;
5. for(i = 0 to n_t){//n_t is the distribution number of the tasks
6. Distribute-tasks-to-slaves($task_i$);
7. If-there-are-overload/underload-nodes(){
8. Generate-forward-ant();
9. Compute-moving-probability();
10. Move to next node;
11. if(node-is-candidate)
12. Generate-backward-ant();
13. Start-timer-for-backward-ant($timer_{na}$);
14. Update-pheromone-by-forward-ant();
15. if($timer_{na} > 0$)
16. Update-pheromone-by-backward-ant();
17. if(task-in-slave-successful)
18. Increase- pheromone;
19. if(task-in-slave-failed)
20. Decrease- pheromone;
21. }
22. if(satisfy-load-balancing){
23. Do-load-balancing();
24. continue;
25. }
26. else if(need-new-tasks)
27. Go to 3;
28. }
29. End of algorithm

5. Simulation and Results Analysis

CloudSim 3.0 [35] is chosen as the simulation toolkit for cloud computing, which is easily used to test and verify the feasibility and stability of the proposed load balancing approach. Windows XP (professional, Microsoft, Los Angeles, CA, US, 2010) OS with 2.53 Ghz CPU, 2 GB of memory, JDK (7.0, Sun Microsystems, Palo Alto, CA, US, 2009) and Ant (1.8.4, Apache Software Foundation, Forest Hill, MD, US, 2011) are employed to run the simulations.

First of all, we test different combinations of the parameters in order to find out the optimal parameters setting. Generally, the CPU has the highest importance in hardware resources [14,16], so we set $\psi_1 : \psi_2 : \psi_3 : \psi_4 : \psi_5 = 12 : 7 : 7 : 7 : 7$. As [34], the coefficient of evaporation for slave node is set $\rho = 0.5$ while the coefficient of evaporation for backward ant is set $\pi = 0.5$. We arrange 10,000 slave

nodes in the cloud platform and each node is set with different processing capabilities randomly. The pheromone initialization of each node is calculated after randomly setting slave nodes. Each iteration is regarded as a unit time in our experiment. In a unit time, a new job with 10,000 tasks is distributed to 10,000 slave nodes at random. Each task can be finished within [1,3] unit time. All ants can only move one edge from one slave node to the other and the pheromone update is carried out once in a unit time.

During the process, 50 slave nodes that the task number is bigger than nine are selected as overload nodes and 50 slave nodes that the task number is smaller than two are selected as underload nodes at random. If there are not enough slave nodes that can satisfy such conditions, the actual number of overload nodes and underload nodes are chosen. The forward ants are generated from both the overload nodes and underload nodes in order to accelerate searching process. The load balancing is performed when reaching the maximum searching step. The maximum search step stands for the maximum distance the ant can reach within one unit time. After load balancing operations, the cloud platform goes into the next unit time and a new job with 10,000 tasks comes.

Table 1 gives the experimental results for the number of iterations (Num) and the convergence time (CT) with diverse parameters when reaching load balancing state. For each group of parameters, we carry out the simulation 20 times and the results of Num and CT are the average values.

Table 1. Number of iterations and convergence time with diverse parameters.

NO	μ	ν	θ	ω	σ	α	β	γ	κ	Num	CT
1	0.1	0.1	±0.1	0.3	0.1	1	1	1	1	334	15.432
2	0.1	0.1	±0.1	0.3	0.1	1	1	2	2	319	16.212
3	0.1	0.1	±0.1	0.3	0.1	2	2	1	1	394	20.134
4	0.1	0.1	±0.1	0.3	0.1	2	2	2	2	405	19.293
5	0.2	0.2	±0.2	0.4	0.2	1	1	1	1	315	14.289
6	0.2	0.2	±0.2	0.4	0.2	1	1	2	2	287	12.785
7	0.2	0.2	±0.2	0.4	0.2	2	2	1	1	376	16.112
8	0.2	0.2	±0.2	0.4	0.2	2	2	2	2	389	17.834
9	0.3	0.3	±0.3	0.2	0.3	1	1	1	1	329	14.115
10	0.3	0.3	±0.3	0.2	0.3	1	1	2	2	301	13.454
11	0.3	0.3	±0.3	0.2	0.3	2	2	1	1	392	16.298
12	0.3	0.3	±0.3	0.2	0.3	2	2	2	2	421	21.103

It can be seen from Table 1 that the result in row 6 has the minimum number of iterations and convergence time to achieve load balancing, which means the presented approach can reach an optimal state by these parameter settings. Thus, this group of parameter settings are used for the following simulations.

Since the number of ants has an impact on the results, we perform the study by using different number of ants with 50 (Ant-50), 100 (Ant-100), and 200 (Ant-200) ants. The maximum searching step for each ant is set to 10. We adopt the degree of load balancing to describe the results, which stands for the balance level of the virtual machine in the cloud platform. The degree of load balancing is defined as:

$$LB = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Load_i - Load_{avg})^2} \tag{20}$$

where $Load_i$ is the load of virtual machine, that is, the load of the slave node, and $Load_{avg}$ indicates the average load of all virtual machines. The bigger the degree of load balancing is, the more unbalanced the load will be.

We limit the number of iterations to below 600 and the simulation results of execution time and degree of load balancing by different numbers of ants are shown in Figures 2 and 3.

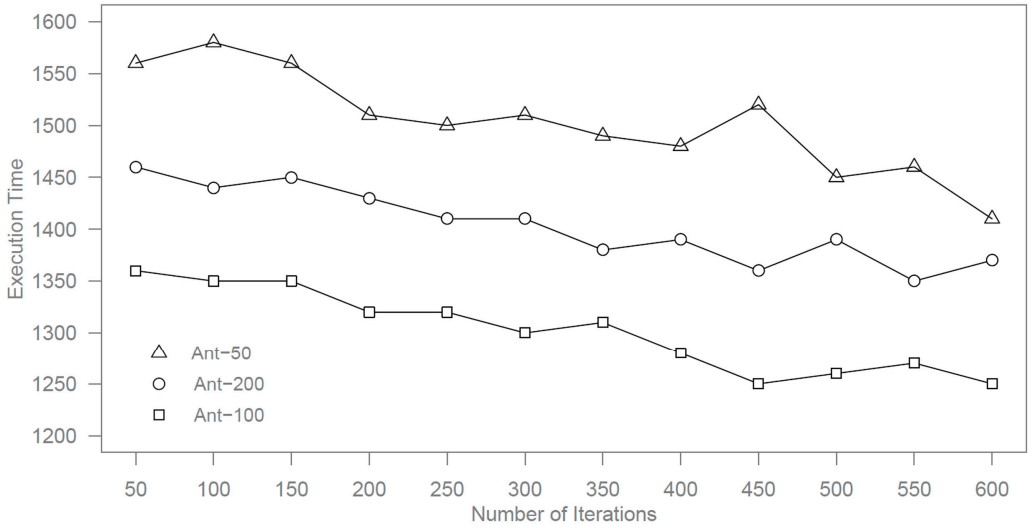

Figure 2. Execution time by different numbers of ants.

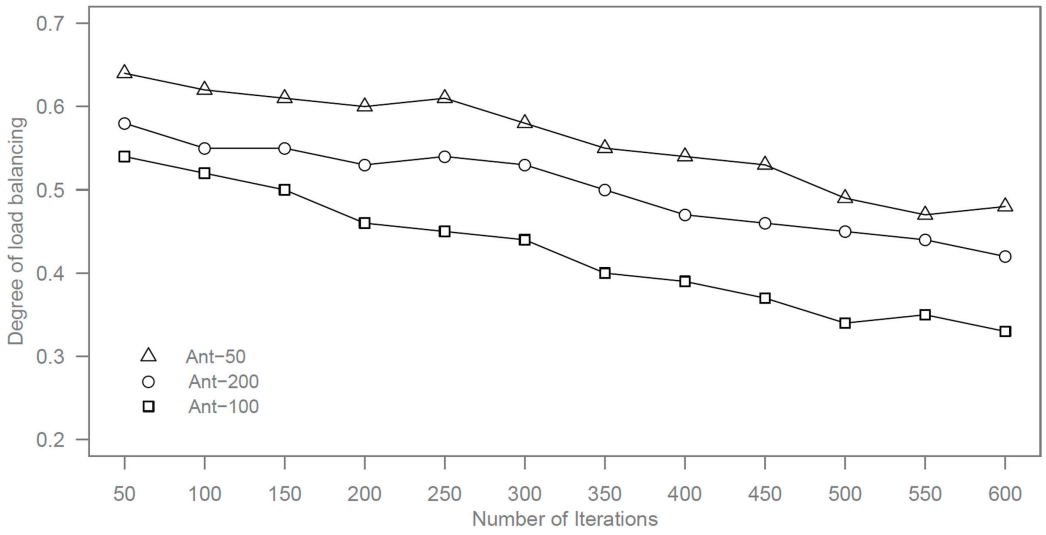

Figure 3. Degree of load balancing by different numbers of ants.

It can be seen from the results that the minimum values are obtained for the number of iterations and the degree of load balancing when the number of ants is set to 100.

Apart from the number of ants, the maximum searching step for each ant also has influence on the results, so we carry out the experiment by different maximum searching steps of five (Step-5), 10 (Step-10), and 20 (Step-20) steps. Figures 4 and 5 shows the results of time execution and degree of load balancing by different maximum searching steps.

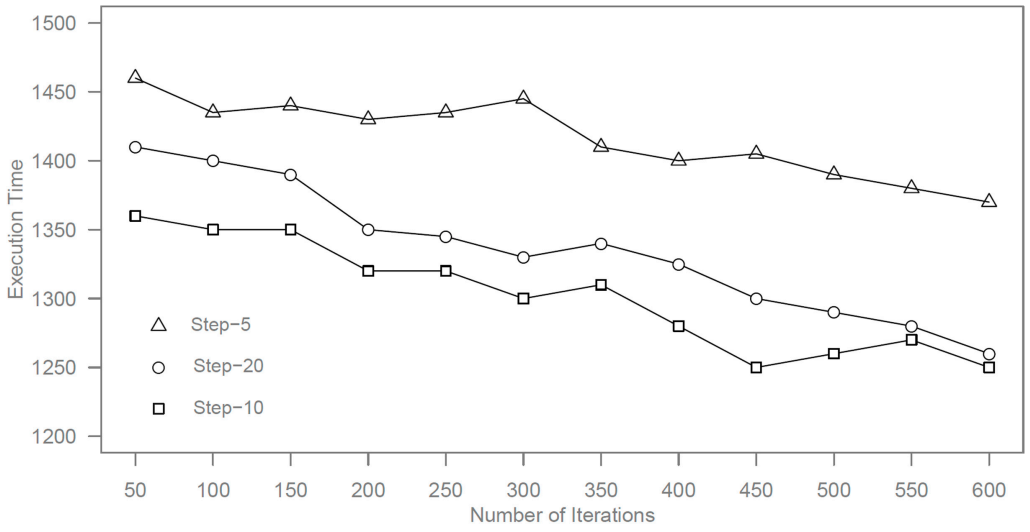

Figure 4. Execution time by different maximum searching steps.

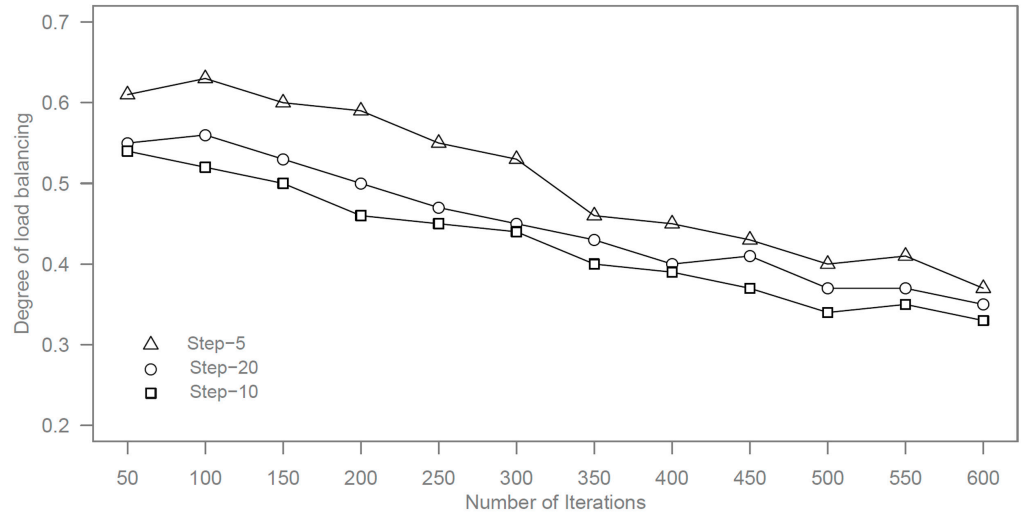

Figure 5. Degree of load balancing by different maximum searching steps.

It can be seen from the results that the minimum searching time is obtained when the maximum searching step is set to 10. Due to the forward ants in step-20 needing to go further than the ants in step-10, it takes more time for forward ants in step-20. For the maximum searching with five steps, the performance is weak because most of forward ants cannot find out the proper slave nodes within five steps.

In addition, we compare the presented approach by improved ant colony optimization (IACO) with random algorithm (RA) and LBVS algorithm [20] for load balancing in cloud platforms. In this experiment, the number of slave nodes is 10,000, and the maximum searching step is 10. Other parameters are set by the optimal values from the above simulations.

The LBVS algorithm was not proposed for cloud computing, so we combine heuristic methods to LBVS for getting better performance in our experiments. For random algorithms, 100 ants are distributed randomly in 100 slave nodes. In a unit time, the ants record the maximum and minimum slave nodes in searching processes and perform load balancing before the next unit time comes. The number of iterations is also limited to 600. The comparison results are shown in Figures 6 and 7.

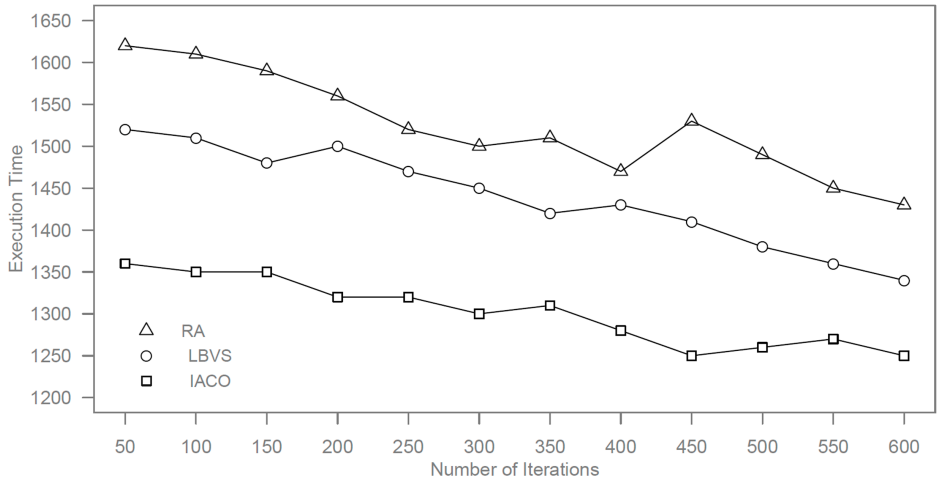

Figure 6. Execution time by different algorithms.

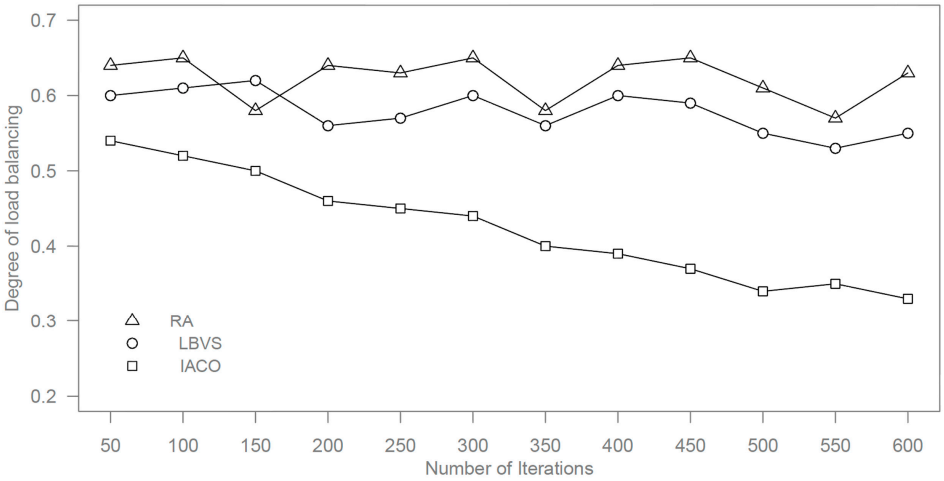

Figure 7. Degree of load balancing by different algorithms.

In order to test the ability for different scale of slave nodes in cloud computing platform, we carry out load balancing experiments with the numbers of slave nodes from 10,000 to 100,000. The results are shown in Figure 8.

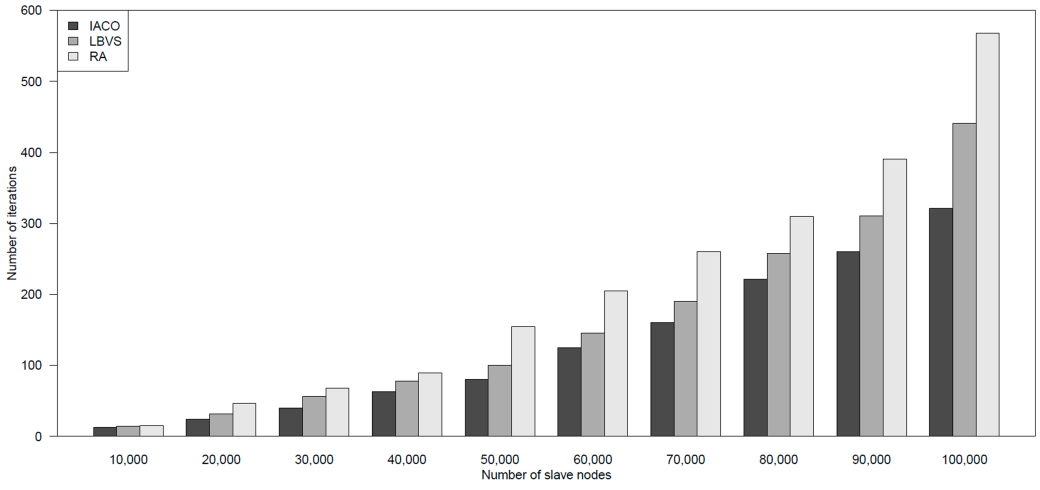

Figure 8. Execution time for load balancing by different algorithms.

It can be seen from the results that the proposed approach has a better speed of convergence and can be used in the cloud computing platform with a huge number of slave nodes. The random algorithm has a large fluctuation and more execution time because it is an unpredictable algorithm. LBVS is better than a random algorithm but it also fluctuates, like the random algorithm.

Therefore, the proposed approach has good efficiency for load balancing in cloud computing, which satisfies the needs of dynamic task allocation in the cloud platform. The proposed approach can make the cloud platform evolve over time and speed up the convergence time. Compared to a random algorithm and LBVS algorithm, it has better performance and is less time consuming.

6. Conclusions and Future Work

Cloud computing is a rapidly evolving field and changing the way a business or activity can operate, which provides more flexibility and less time to deploy a project for us. Load balancing is one of the key challenges in cloud computing. In order to optimize resource allocation and ensure quality of service, this paper proposed a novel approach for dynamic load balancing based on the improved ant colony optimization. Two dynamic load balancing strategies were applied with the forward-backward ant mechanism and max-min rules. According to the characteristics of cloud computing, we redefined and improved the ant colony optimization by pheromone initialization and pheromone update. By means of such improvements, the speed for searching candidate nodes in load balancing operations can be greatly accelerated. Two kinds of moving rules for the forward ant were given to update the pheromone so as to speed up the convergence and the detailed dynamic load balancing algorithm was also described. Several simulations were illustrated by the improved approach in a cloud computing platform. The results showed that the proposed approach is feasible and effective on load balancing in cloud computing and also has better performance than a random algorithm and LBVS algorithm.

In future work we will further study the triggering mechanism for ant generation and the strategy for pheromone update in order to significantly reduce the searching time for candidate nodes. Furthermore, we will investigate how to introduce other intelligent algorithms into our approach, with the purpose of improving system performance and efficiency.

Acknowledgments

This paper is supported by the Natural Science Foundation of Hubei Province (No. 2013CFC009).

Author Contributions

Ren Gao and Juebo Wu designed experiments and carried out experiments. Ren Gao analyzed the data results and Juebo Wu contributed materials. The manuscript was written by both Ren Gao and Juebo Wu.

Conflicts of Interest

The authors declare no conflict of interest.

References

1. Wang, L.; von Laszewski, G.; Younge, A.; He, X.; Kunze, M.; Tao, J.; Fu, C. Cloud computing: A perspective study. *New Gener. Comput.* **2010**, *28*, 137–146.
2. Armbrust, M.; Fox, A.; Griffith, R.; Joseph, A.D.; Katz, R.; Konwinski, A.; Lee, G.; Patterson, D.; Rabkin, A.; Stoica, I.; *et al*. A view of cloud computing. *Commun. ACM* **2010**, *53*, 50–58.
3. Zhang, Q.; Cheng, L.; Boutaba, R. Cloud computing: State-of-the-art and research challenges. *J. Internet Serv. Appl.* **2010**, *1*, 7–18.
4. Lee, Y.C.; Zomaya, A.Y. Energy efficient utilization of resources in cloud computing systems. *J. Supercomput.* **2010**, *60*, 268–280.
5. Rimal, B.P.; Choi, E.; Lumb, I. A taxonomy and survey of cloud computing systems. In Proceedings of the Fifth International Joint Conference on INC, IMS and IDC, 2009. NCM'09, Seoul, Korea, 25–27 August 2009; pp. 44–51.
6. Nishant, K.; Sharma, P.; Krishna, V.; Gupta, C.; Singh, K.P.; Nitin, N.; Rastogi, R. Load Balancing of Nodes in Cloud Using Ant Colony Optimization. In Proceedings of the 2012 UK Sim 14th International Conference on Computer Modelling and Simulation (UKSim), Cambridge, UK, 28–30 March 2012; pp. 3–8.
7. Zhang, Z.; Zhang, X. A load balancing mechanism based on ant colony and complex network theory in open cloud computing federation. In Proceedings of the 2010 2nd International Conference on Industrial Mechatronics and Automation (ICIMA), Wuhan, China, 30–31 May 2010; Volume 2, pp. 240–243.
8. Mishra, R.; Jaiswal, A. Ant colony optimization: A solution of load balancing in cloud. *Int. J. Web Semant. Technol.* **2012**, *3*, 33–50.
9. Sesum-Cavic, V.; Kuhn, E. Comparing configurable parameters of swarm intelligence algorithms for dynamic load balancing. In Proceedings of the 2010 Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshop (SASOW), Budapest, Hungary, 27–28 September 2010; pp. 42–49.
10. Sesum-Cavic, V.; Kuhn, E. Applying swarm intelligence algorithms for dynamic load balancing to a cloud based call center. In Proceedings of the 2010 4th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO), Budapest, Hungary, 27 September 2010; pp. 255–256.
11. Liu, Z.; Wang, X. A PSO-Based Algorithm for Load Balancing in Virtual Machines of Cloud Computing Environment. In *Advances in Swarm Intelligence*; Lecture Notes in Computer Science; Tan, Y., Shi, Y., Ji, Z., Eds.; Springer: Berlin, Germany; Heidelberg, Germany, 2012; Volume 7331, pp. 142–147.
12. Feng, X.; Pan, Y. DPSO Resource Load Balancing in Cloud Computing. *Comput. Eng. Appl.* **2011**, *11*, 1–8.
13. Dorigo, M. Optimization, Learning and Natural Algorithms. Ph.D. Thesis, Politecnico di Milano, Milan, Italy, May 1992.
14. Thakur, S.; Tripathi, S. *Load Balancing in a Network using Ant Colony Optimization Technique*; National Institute of Technology Rourkela: Rourkela, India, 2009.

15. Sim, K.M.; Sun, W.H. Ant colony optimization for routing and load-balancing: Survey and new directions. *IEEE Trans. Syst. Man Cybern. Part A* **2003**, *33*, 560–572.
16. Keskinturk, T.; Yildirim, M.B.; Barut, M. An ant colony optimization algorithm for load balancing in parallel machines with sequence-dependent setup times. *Comput. Oper. Res.* **2012**, *39*, 1225–1235.
17. Bhadani, A.; Chaudhary, S. Performance evaluation of web servers using central load balancing policy over virtual machines on cloud. In Proceedings of the Third Annual ACM Bangalore Conference, Bangalore, India, 22–23 January 2010; pp. 16–19.
18. Hu, J.; Gu, J.; Sun, G.; Zhao, T. A scheduling strategy on load balancing of virtual machine resources in cloud computing environment. In Proceedings of the 2010 Third International Symposium on Parallel Architectures, Algorithms and Programming (PAAP), Dalian, China, 18–20 December 2010; pp. 89–96.
19. Ni, J.; Huang, Y.; Luan, Z.; Zhang, J.; Qian, D. Virtual machine mapping policy based on load balancing in private cloud environment. In Proceedings of the 2011 International Conference on Cloud and Service Computing (CSC), Hong Kong, China, 12–14 December 2011; pp. 292–295.
20. Zhao, Y.; Huang, W. Adaptive Distributed Load Balancing Algorithm based on Live Migration of Virtual Machines in Cloud. In Proceedings of the Fifth International Joint Conference on INC, IMS and IDC, 2009. NCM'09, Seoul, Korea, 25–27 August 2009; pp. 170–175.
21. Ren, X.; Lin, R.; Zou, H. A dynamic load balancing strategy for cloud computing platform based on exponential smoothing forecast. In Proceedings of the 2011 IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS), Beijing, China, 15–17 September 2011; pp. 220–224.
22. Wang, S.C.; Yan, K.Q.; Liao, W.P.; Wang, S.S. Towards a Load Balancing in a three-level cloud computing network. In Proceedings of the 2010 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT), Chengdu, China, 9–11 July 2010; Volume 1, pp. 108–113.
23. Liu, H.; Liu, S.; Meng, X.; Yang, C.; Zhang, Y. LBVS: A load balancing strategy for virtual storage. In Proceedings of the 2010 International Conference on Service Sciences (ICSS), Hangzhou, China, 13–14 May 2010; pp. 257–262.
24. Wu, T.-Y.; Lee, W.-T.; Lin, Y.-S.; Lin, Y.-S.; Chan, H.-L.; Huang, J.-S. Dynamic load balancing mechanism based on cloud storage. In Proceedings of the Computing, Communications and Applications Conference (ComComAp), Hong Kong, China, 11–13 January 2012; pp. 102–106.
25. Bo, Z.; Ji, G.; Jieqing, A. Cloud Loading Balance algorithm. In Proceedings of the 2010 2nd International Conference on Information Science and Engineering (ICISE), Hangzhou, China, 4–6 December 2010; pp. 5001–5004.
26. Randles, M.; Lamb, D.; Taleb-Bendiab, A. A comparative study into distributed load balancing algorithms for cloud computing. In Proceedings of the 2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops (WAINA), Perth, Australia, 20–23 April 2010; pp. 551–556.
27. Jaspreet, K. Comparison of load balancing algorithms in a Cloud. *Int. J. Eng. Res. Appl.* **2012**, *2*, 1169–1173.
28. Shaveta, N.; Raj, G. Comparative Analysis of Load Balancing Algorithms in Cloud Computing. *Int. J. Adv. Res. Comput. Eng. Technol.* **2012**, *1*, 120–124.

29. Nakrani, S.; Tovey, C. On honey bees and dynamic server allocation in Internet hosting centers. *Adapt. Behav.* **2004**, *12*, 223–240.
30. Sivanandam, S.N.; Visalakshi, P. Dynamic task scheduling with load balancing using parallel orthogonal particle swarm optimisation. *Int. J. Bio-Inspired Comput.* **2009**, *1*, 276–286.
31. Visalakshi, P.; Sivanandam, S.N. Dynamic Task Scheduling with Load Balancing using Hybrid Particle Swarm Optimization. *Int. J. Open Probl. Compt. Math.* **2009**, *2*, 475–488.
32. Ludwig, S.A.; Moallem, A. Swarm intelligence approaches for grid load balancing. *J. Grid Comput.* **2011**, *9*, 279–301.
33. Ali, A.; Belal, M.A.; al-Zoubi, M.B. Load Balancing of Distributed Systems Based on Multiple Ant Colonies Optimization. *Am. J. Appl. Sci.* **2010**, *7*, 433–438.
34. Di Caro, G.; Dorigo, M. Mobile agents for adaptive routing. In Proceedings of the Thirty-First Hawaii International Conference on System Sciences, Kohala Coast, HI, USA, 6–9 January 1998; Volume 7, pp. 74–83.
35. Calheiros, R.N.; Ranjan, R.; Beloglazov, A.; de Rose, C.A.F.; Buyya, R. CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software* **2011**, *41*, 23–50.