Chapter 111
Bidirectional Ant Colony Optimization Algorithm for Cloud Load Balancing

Shin-Hung Li and Jen-Ing G. Hwang

Abstract Cloud computing supplies convenient, on-demand network access to shared computing resources. Although cloud computing has many advantages, some core issues, such as load balancing, need to be resolved. Cloud load balancing provides a mechanism to allocate the appropriate workload for each virtual machine in a cloud computing environment. One of the main characteristics of the cloud computing environment is rapid change; therefore, a good load-balancing method should adapt quickly to environment changes to achieve high performance. This paper proposes a load-balancing method based on the Ant Colony Optimization (ACO) algorithm and describes a simulation using the CloudSim toolkit package. Experimental results show that the proposed method is an effective ACO method for cloud balancing.

Keywords Cloud computing · Load balancing · Ant colony optimization · Pheromones

111.1 Introduction

With the increasing popularity of cloud computing, the cloud faces a major challenge in managing an extremely large number of service requests. In order to maintain the stability of the system and provide good service, an effective load-balancing technique for task scheduling in dynamic cloud environment is required.

S.-H. Li · J.-I. G. Hwang (✉)
Department of Computer Science and Information Engineering, Fu Jen Catholic University, Taipei 24205, Taiwan, Republic of China
e-mail: jihwang@csie.fju.edu.tw

S.-H. Li
e-mail: 400226064@mail.fju.edu.tw

J. Juang et al. (eds.), *Proceedings of the 2nd International Conference on Intelligent Technologies and Engineering Systems (ICITES2013)*, Lecture Notes in Electrical Engineering 293,
DOI: 10.1007/978-3-319-04573-3_111,

Some researchers developed effective algorithms based on the behavior of social animals, because social organisms can effectively adapt to a changing environment. For example, an ant colony can dynamically allocate tasks without a central manager. The Ant Colony Optimization (ACO) algorithm has been used to solve the load-balancing problem in Peer-to-Peer (P2P) environments [1, 2], and in grid computing [3–5]. Recently, ACO has also been applied to the cloud environment [6–8].

In this paper, an improved ACO algorithm is proposed for load balancing in the cloud environment and the CloudSim tool [9, 10], is used to conduct simulations for comparing the effectiveness of different load-balancing strategies.

The rest of the paper is organized as follows: Sect. 111.2 briefly describes the simple ACO algorithm. Section 111.3 discusses the proposed bidirectional ACO algorithm. Section 111.4 shows the experimental results. Finally, Sect. 111.5 presents conclusions.

111.2 Simple Ant Colony Optimization

Ant colony algorithms are computational swarm intelligence methods that were inspired by a number of different behaviors observed in ant colonies. A simple ACO algorithm was proposed by Dorigo et al. [11, 12].

Dorigo et al.'s ACO algorithm was based on the foraging behavior of ants while searching for the shortest path between the ant nest and a food source. Likewise, one can consider the general problem as finding the shortest path between two nodes on a graph. During the shortest-path search, cooperation among the ants is achieved by releasing pheromones to influence the behavior of other ants. How forager ants decide which edge to follow is a probabilistic technique and depends on the pheromone concentrations on different edges. Let τ_{ij} be the pheromone concentration between nodes of i and j in the graph. Initially, the value $\tau_{ij}(0)$ is assigned a random value to indicate the pheromone at time zero, and a number of ants are placed at the starting node. Each ant then creates a path between the starting node and the terminating node by determining the next edge from the current node at every time step (iteration) t. The transition probability $p_{ij}^{k}(t)$ of the kth ant moving from node i to node j at time step t is defined as

$$p_{ij}^{k}(t) = \begin{cases} \dfrac{(\tau_{ij}(t))^{\alpha}}{\sum_{u \in J_k(i)} (\tau_{iu}(t))^{\alpha}} & \text{if} \quad j \in J_k(i) \\ 0 & \text{otherwise} \end{cases}, \tag{111.1}$$

where $J_k(i)$ is the set of feasible nodes linked to node i along the path of the kth ant, and α is a parameter used to intensify the influence of pheromone concentrations. When an ant has traversed a path within the time step t, the ant retraces its path and deposits pheromone on every edge of the path. The concentration of the deposited

pheromone on the edge (i, j) is inversely proportional to the length of the path. That is,

$$\Delta\tau_{ij}^{k}(t) \propto \frac{1}{L^{k}(t)} \tag{111.2}$$

where $\Delta\tau_{ij}^{k}$ is the deposited pheromone, and $L^{k}(t)$ is the length of the path created by the kth ant at time step t. The total pheromone intensity of an edge is then updated using Eq. (111.3).

$$\tau_{ij}(t+1) = (1-\beta_{eva}) \times \tau_{ij}(t) + \sum_{k=1}^{n} \Delta\tau_{ij}^{k}(t) \tag{111.3}$$

where β_{eva} is the pheromone evaporation rate with $\beta_{eva} \in [0, 1]$, and n is the number of ants. The iterations are executed until one of the terminating conditions is true. The solution is the shortest path found among all the paths traversed by the ants.

111.3 The Proposed Algorithm

In this paper, a Bidirectional Load Balancing ACO (BLBACO) algorithm that is inspired by Li et al. [6] and Nishant et al. [7] is proposed. The BLBACO used the Load Balancing ACO (LBACO) [6] characteristics to determine the loading and computing capacity of each virtual machine, and BLBACO also adapted the concept proposed in [7], where the movements of the ants in the system are defined as either a forward movement or a backward movement. Forward movement is the direction of ants moving from the nest to a food source, and backward movement is the direction from the food source to the nest. In the computing cloud, forward movement can be defined as the direction of an under-loaded virtual machine (nest) seeking an over-loaded virtual machine (food), while backward movement is the reversed direction. Once the search is completed, a portion of the tasks that were originally assigned to the over-loaded virtual machine can be reassigned to the under-loaded virtual machine.

The proposed algorithm performs the following procedures:

- Initialize pheromone

At the beginning, an amount of pheromone intensity is initialized at each virtual machine, and each ant is randomly positioned on a virtual machine.

- Select a virtual machine

The probability of the kth ant selecting the jth virtual machine is calculated by the following formula [6]:

$$p_j^k(t) = \begin{cases} \dfrac{[\tau_j(t)]^{\alpha}[EV_j]^{\beta}[LB_j]^{\gamma}}{\sum_{u \in J_k(i)} [\tau_u(t)]^{\alpha}[EV_u]^{\beta}[LB_u]^{\gamma}} & , if \ \in 1 \ldots m \\ 0 & , otherwise \end{cases} \tag{111.4}$$

where $p_j^k(t)$ is the probability of the kth ant selecting the virtual machine j at time step t, $\tau_j(t)$ is the pheromone concentration deposited on the virtual machine j at time step t,EV_j is the computing capacity of the virtual machine j, LB_j is the load balancing factor of the virtual machine j, α, β, and γ are parameters, and m is the total number of virtual machines.

- Update pheromones

Two sets of formulae are used to update pheromones. If all virtual machines are over-loaded or all virtual machines are under-loaded, a one-way update formula is used; otherwise, a set of bidirectional update formula is used. The formulae are described as follows:

Case 1: If all virtual machines are over-loaded state or all virtual machines are under-loaded, a one-way formula is used to update the pheromone intensity:

$$\tau_j(t+1) = (1 - \beta_{eva}) \times \tau_j(t) + \sum_{k=1}^{n} \Delta\tau_j^k(t) \tag{111.5}$$

where $\tau_j(t)$ is the total pheromone intensity of node j at time step t, $\Delta\tau_j^k$ is the pheromone deposited by ant k, and β_{eva} is the pheromone evaporation rate as defined in Eq. (111.3).

Case 2: If the cloud environment contains both over-loaded and under-loaded virtual machines, two pheromone-update formulae are used [7]. The first formula is used when an ant is at an under-loaded node and is searching for an over-loaded node:

$$FP_j(t+1) = (1 - \beta_{eva})FP_j(t) + \sum_{k=1}^{n} \Delta FP_j^k \tag{111.6}$$

where $FP_j(t)$ is the total amount of foraging pheromone at node j at time step t, and ΔFP_j^k is the pheromone deposited by ant k.

The other formula is used when an ant is at an over-loaded node and is returning to the under-loaded node:

$$TP_j(t+1) = (1 - \beta_{eva})TP_j(t) + \sum_{k=1}^{n} \Delta \mathrm{T}P_j^k \tag{111.7}$$

Table 111.1 Number of virtual machines = 50 (fixed)

Iteration	Algorithm	500 tasks Makespan (ms)	300 tasks	100 tasks	50 tasks	25 tasks
1	LBACO	93.86	83.27	60.55	35.17	22.13
	BLBACO	87.77	77.98	54.34	24.48	20.38
30	LBACO	82.11	73.18	52.95	27.34	18.39
	BLBACO	76.09	69.38	45.71	19.11	16.80
70	LBACO	74.62	68.75	48.17	24.24	15.76
	BLBACO	69.78	65.25	42.81	16.02	14.43
100	LBACO	64.0	58.91	39.69	20.15	11.72
	BLBACO	58.10	55.86	36.15	11.32	10.96

Table 111.2 Number of virtual machines = Number of tasks

Iteration	Algorithm	500 tasks Makespan (ms)	300 tasks	100 tasks	50 tasks	25 tasks
1	LBACO	52.64	48.55	48.85	46.44	47.70
	BLBACO	38.62	34.83	35.05	34.84	36.37
30	LBACO	46.58	38.88	43.19	40.74	41.71
	BLBACO	34.18	28.57	30.22	29.08	31.05
70	LBACO	40.01	35.29	37.49	37.44	36.39
	BLBACO	32.25	26.44	27.688	26.33	28.46
100	LBACO	34.75	31.88	31.15	31.64	30.30
	BLBACO	28.028	22.83	23.82	22.59	23.95

where $TP_j(t)$ is the total amount of tracing pheromone of node j at time step t, and ΔTP_j^k is the pheromone deposited by ant k.

111.4 Experimental Results

The proposed BLBACO algorithm was implemented using the Java language and was tested on the CloudSim platform [9, 10]. The LBACO [6] algorithm has outperformed the First-Come-First-Serve (FCFS) algorithm and the basic ACO algorithm. In addition, the approach proposed by Nishant et al. [7] is not suitable for a dynamic environment. Therefore, the proposed BLBACO algorithm compared with LBACO [6]. The experiments used the parameter settings in [6]; that is, the number of ants n is 8, the pheromone evaporation rate β_{eva} is 0.01, and the parameters α, β, and γ in Eq. (111.4) are 3, 2, and 8, respectively.

In the first category of experiments, we fixed the total number of virtual machines to be 50 in each trial, and we varied the number of task requests to be 500, 300, 100, 50, and 25 in different trials. BLBACO and LBACO each executed 5 times for each trial. The makespan averages are shown in Table 111.1.

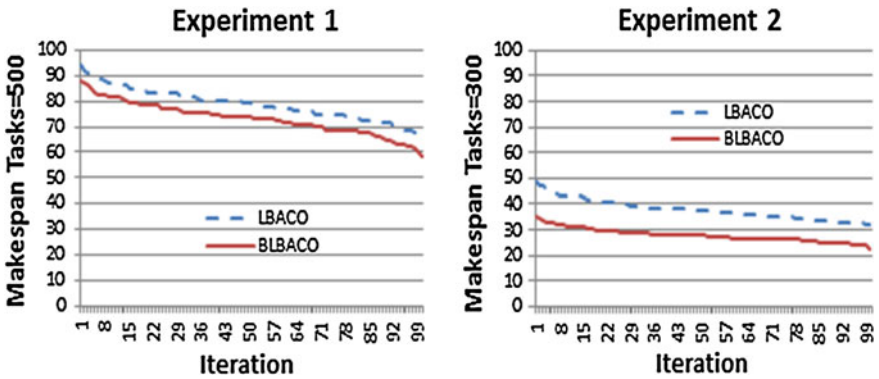

Fig. 111.1 Overall performance of the first experiment category with 500 tasks (*left*) and of the second category experiment with 300 tasks (*right*)

The second category of experiments verified the effectiveness of load-balancing strategies when the cloud environment contains both over-loaded and under-loaded virtual machines. To simplify the experiments, we set an equal number of virtual machines and task requests, and we assumed that the system would contain both under-loaded and over-loaded nodes. The results are listed in Table 111.2. To easily compare the performance of BLBACO with the performance of LBACO, we choose one trial from each category of experiments. The left panel of Fig. 111.1 shows the performance of the first category experiment with 50 virtual machines and 500 task requests, and the right panel shows the performance of the second category experiment with 300 virtual machines and 300 task requests. Based on the experimental results, the performance of BLBACO is better than that of LBACO. This validates the effectiveness of the proposed BLBACO algorithm.

111.5 Conclusions

In this paper, an improved method of cloud load balancing based on the ACO algorithm was proposed. The proposed BLBACO algorithm considers the loading and computing capacity of each virtual machine with bidirectional movement in ACO. The experiments showed that the BLBACO algorithm has better performance compared to LBACO [6]. This indicates that BLBACO can effectively deal with task allocation in a dynamic cloud environment.

References

1. Montresor, A., Meling, H., & Babaoglu, O. (2002). Messor: Load-balancing through a swarm of autonomous agents. In *AP2PC* (pp. 125–137).
2. Michlmayr, E. (2006). Ant algorithms for search in unstructured peer-to-peer networks. In *22th IEEE International Conference on Data Engineering Workshops*. Atlanta, GA, USA.
3. Salehi, M. A., Deldari, H. (2006). Grid load balancing using an echo system of intelligent ants. In *24th IASTED Parallel and Distributed Computing and networks*.
4. Fidanova, S., & Durchova, M. (2006). Ant Algorithm for Grid Scheduling Problem. In I. Lirkov, S. Margenov, & J. Waśniewski (Eds.), *LSSC 2005. LNCS* (Vol. 3743, pp. 405–412). Heidelberg: Springer.
5. Lee, Y. H., Leu, S., & Chang, R. S. (2011). *Improving job scheduling algorithms in a grid environment* (Vol. 27, pp. 991–998). Amsterdam: Future generation computer systems, FGCS.
6. Li, K., Xu, G., Zhao, G., Dong, Y., & Wang, D. (2011). Cloud task scheduling based on load balancing ant colony optimization. In *Conference of ChinaGrid* (pp. 3–9). Liaoning.
7. Nishant, K., Sharma, P., Krishna, V., Gupta, C., Singh, K. P., Nitin, N., & Rastogi, R. (2012). Load balancing of nodes in cloud using ant colony optimization. In *14th International Conference on Conference of Modelling and Simulation*. Cambridge.
8. Kansal, N. J. (2012). Existing load balancing Techniques in cloud computing: A systematic review. *Journal of Information Systems and Communication*.
9. Wickremasinghe, B., Calheiros, R. N., & Buyya, R. (2010). CloudAnalyst: A cloudsim based visual modeller for analysing cloud computing environments and applications. In *24th IEEE International Conference on Advanced Information Networking and Applications* (pp. 446–452). Perth, WA.
10. Buyya, R., Ranjan, R., & Calheiros, R. N (2010) CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. In *IEEE International Conference on High Performance Computing* (Vol. 41, pp. 23–50).
11. Dorigo, M., & Di Caro, G. (1999). Ant colony optimization: A new meta-heuristic. In: D. Corne, M. Dorigo, & F. Glover (Eds.), *New ideas in optimization* (pp. 11–32).
12. Dorigo, M., & Stützle, T. (2001). An experimental study of the simple ant colony optimization algorithm. In *WSES International Conference on Evolutionary Computation* (pp. 253–258).