Load Balancing Strategy of Cloud Computing based on Artificial Bee Algorithm

Jing Yao*, Ju-hou He

* Dept. of Computer Science Shaanxi Normal University Xi'an, China, ruirui8718@163.com
2 Dept. of Computer Science Shaanxi Normal University Xi'an, China, juhouh@gmail.com

Abstract

It is indispensable to cloud computing that implements resource access by load balancing. According to the characteristics and requirements of cloud computing environments, an artificial bee colony algorithm (ABC) has been presented in this paper. Hundreds of thousand of simultaneous requests with the same type queued in the same server for the original ABC algorithm. Consequently, the local resource-intensive phenomenon arised and deteriorated load balancing. Due to the failure of this mechanism for the above case, an improved ABC is proposed. By replacing other types of requests with the next served request, the type of request is changed. It ended the accumulation of request and improved the system throughput. Experimental results show that ABC algorithm-based load balancing mechanism is applause for its stability and the improved ABC does well in the scalability.

Keywords: Cloud computing, load balance, Artificial Bee Colony algorithm (ABC).

1. Introduction

Cloud computing is an Internet-based computing mode, which could distribute tasks across a large number of resource pools composed by computers. Consequently, all kinds of applications are able to obtain enough computing power, storage space and information services [1]. With the help of cloud computing, the resource of software and hardware could be shared reasonably to avoid the shortcomings of information redundancy occurred in early distributed network. However, the resources sharing will lead to a range of issues. Such as multiple concurrent service requests towards the server at the same time may cause it to standstill, while other servers are still idle. This is known as the collapse caused by system load imbalance. Thus, under the cloud computing environment, one of the urgent problems is the distributed control. As a self-organized network, the basic function of distributed control is to know the operational status of each server and resource usage through the information exchange between the local and the whole network, and to allocate the resources reasonably for new service requests. Nowadays, cloud computing load balancing has become one of the hot issues need to be resolved. Furthermore, there are many literatures [2][3] focusing on homogeneous environment load balancing, but under the cloud computing environment, the system of homogeneous server is less. Amazon Elastic Compute Cloud EC2 gives a good example in this area. Its dynamic load balancing is completed by replicating instances of the specific middleware platform for Web services. So as for the heterogeneous environment, the middleware approach can be used to deliver information. In addition, the size of these systems could not be ignored as well. If the system collapse is due to server maintenance, it will bring a huge loss to the cloud users. So, how to deal with the requests in the dynamic system is also an important issue [4]. Based on this, literature [5] presents a self-organizing, self-managing, self-protecting, self-adjusting load-balancing mechanism which has the advantage of adjusting the balance algorithm flexibly for different type of environment. Literature [6] presents a load-balancing mechanism for wide range distributed systems. This algorithm is so simple as to reduce the average system response time and improve the system QOS values. It uses biased random sampling. When a request comes in, a server is randomly selected with certain probability, then in-degree of node is reduced. The decrease of in-degree stands for the reduction of resources available. When the task is completed, adding a new entry side to the server (node). Therefore, the condition to achieve load balancing is that each node has more or less number of the entry sides.

In summary, the load balancing mechanism in cloud computing environments requires three conditions as follows:
When the local load case is not very heavy, it can make the local self-organization in order to reduce the information exchange [7].

The cloud computing balancing mechanism could be used in the heterogeneous environment.

In order not to affect the system average response time, cloud computing load balancing mechanism should increase the system throughput as much as possible. (This is the most basic demand of load balancing)

Based on the three considerations above, we can see the cloud computing system has a certain similarity with a colony of honeybees foraging and harvesting food. First of all, the system target of cloud computing is consistent with the bees foraging honey. In other words, cloud computing needs larger system throughput, while the bees need nectar as much as possible; Secondly, the honey bee’s gathering mechanism is similar to the system resources requests agents for individual users in cloud computing; Furthermore, each flower to be taken is similar to a server resource in the cloud, and a garden to a services cluster. Therefore, this paper discusses the artificial bee algorithm and applies it to the load-balancing mechanism. Experiments show that our work breaks the limitations of artificial bee algorithm, and make some improvements by increasing the system throughput.

2. Artificial bee algorithm mechanism

2.1. Introduction of artificial bee algorithm mechanism

Artificial bee mechanism, a method of bionics, proposes an optimization method based on the gathering behaviour of honeybees. The smallest searching model of wisdom generated in bees contains three elements: Food source, employed bees and unemployed bees. Two basic behavioural models are: Foraging nectar in the food source and giving up a food source. A bee makes a decision on foraging based on various factors. For instance, its distance to the source of food, the quantity and quality of nectar etc. These constitute the evaluation for food source. All of these evaluation factors will be represented as the profit.

To classify the bees, the employed bees (also known as the leader bee (Leader)) correspond to the food sources one by one. The leader has information of food source it has foraged (such as the distance, direction, the quantity and quality of food source.). Then it shares these information with other bees with a certain probability. While there is another group of unemployed bees. They can be subdivided into two categories named as detecting bees (Detector) and following bees (Follower). The detectors are responsible for searching new food sources near the hives, while the followers are waiting in the hives. By using the leaders’ information they find food. In general, the average number of detectors is 5%-20% of the whole buzzers.

During the process of bees wisdom formation, the exchange of information between the honeybees is the most important part. The dance district is located in a distributed shared-space advert board. The leaders share the information of food source with other bees through the dance (waggle dance). The profit of food source is represented as the duration of dance performance, so the followers observe a large number of bees dance, and analyse the profit to determine where to forage the honey. As the profit is in proportion to the possibility of foraging the food source, the possibility for bees to be recruited is proportional to profit of a particular food source.

The process of the entire system is: initialization, the bees will start searching as scouts, and the search process is completely random. If the bees find the food source, by using their own storage ability, they will record the location information and the profit. And then they begin to gather honey. At this time, the bees will become “The employed”. When completing the gathering, the honey bees return to the hive and remove the honey, and then they will have the following choices:

1. Abandon the food source to be the unemployed bees.
2. Waggle dance in the dance area to recruit more bees to the corresponding food source, and then getting back to the food source to gather honey.
3. Continuing to gather honey in the same food source rather than exchange the information in the dance area.

The staying unemployed Bees have the following choices:

1. Transforming into detectors, and searching for new food source near the hive. Their searching is completely random.
2. After observing the waggle dance in the dance area, with the analysis of the profit they work as following bees, follow the leader bees to reach the corresponding food source, and then began to search its neighborhood and gather honey.
2.2. Artificial colony search algorithm

The gathering honey mechanism above is corresponding to the search algorithm as follows:

(1) Calculating the profit of certain requests (these services will be treated as leader Bees). Profit can be set as needed. Generally calculating the request takes the time of the waiting and CPU time.

(2) Recording the leader bees’ profit into the database. And this database is equivalent to dancing area. The remaining part of the requests (the following bees) can search for information from the database and make a suitable choice after the comparison. Selection method is as the following formula:

If

\[ \text{If the } j\text{-th request has relationship with the } i\text{-th leader bee, sorting the } j\text{-th request into the server queue directly which provides the service to } i\text{-th request before.} \]

Else

\[ \text{calculate the value of } \min S_j, \text{ the formula is as followed:} \]

\[ \min S_j = \min \left( \omega_1 \frac{T_{\text{wait}}} {T_{\text{cpu}}}, + \omega_2 d_j \right) \]

\[ i = 1, 2, \cdots \quad j = 1, 2, \cdots \]

where \( S_j \) is performed as the fitness value for the server of i-th leader bee service to the j-th request. \( \omega_1 \) and \( \omega_2 \) are weight ratio, while \( \omega_1 + \omega_2 = 1 \). \( T_{\text{cpu}} \) is the time taken to run in the CPU for i-th leading bee. \( T_{\text{wait}} \) is the waiting time before the i-th lead bee being served. \( d_j \) presents the hops from the server for which the i-th leading bee serves to the j-th request and \( d_j \leq 3 \). (Even if the server is idle, the increase of hops will consume much time for the request to service and the results taken back.)

Numbering the best server (The i-th corresponding server in \( \min S_j \)) as \( S_j \) for the j-th request.

The remaining part of the request (neither the leader bees nor the following bee) can be assigned to the server queue which is close to it randomly, and make it search higher profit servers. When this part of the service request is finished, we still need to record the CPU-time and waiting time for this request into the database for future reference.

3. Improved artificial colony mechanism

The previous load-balancing mechanisms only take into account lightly loaded node. Furthermore, it has to process a lot of requests, such as the newly arrived and others which came from heavily loaded nodes. And then the load imbalance happens again. In addition, it was discovered that implementation of the honeybee-foraging distributed load balancing solution at the application layer caused a particular topology to emerge at the resource layer [8]. This was reflected as a small number of services attracting a disproportionate amount of connectivity from cooperating services while most services have only a little link. Furthermore, when selecting the server queue for the following bees happens, the first observation is that whether they are associated with some waggle dancing bee in the dancing area. If it is true, they will go to the server queue directly for which the bee in the dancing area served before. However, it is difficult to judge the degree of close contact between requests. It results in a few relationships between a small numbers of requests are all assigned to the same server queue by system, so an imbalance structure comes out. Therefore, in order to overcome the two shortcomings above, the improved colony mechanism will analyse the cloud resources so that each node within the system will be involved in the iterative process. In the view of the quality of service for the whole system, it is necessary for us to take a relatively simple operation to adjust the alternative for the second shortcoming, and then to achieve the load balancing. The differences between the improved load balancing mechanism and honeybee algorithm are as follows:
(1) When the length of a server queue exceeds the predetermined threshold value, the request being served (denoted the request as R1, and server which copes with R1 as S1) will establish contacts with a randomly selected request (denoted as R2) in an adjacent server, and the R1 and R2 are different types. It is also to say that there are no connections between them.

(2) Inserting R2 into the head of the S1 server queue and dealing it with S1 in the next. So when R2 request is completed, its record type will be different with the prior in the database. It prevents new entrants with the same type of R1 to line in the service queue.

(3) Removing R2 from its original queue.

4. Experiment verification and result analysis

For the purpose of comparing the described algorithms above, a series of simulation are conducted in Repast.NET. Given the instability of the algorithm, the simulations were repeated 20 times each. And the experimental data is obtained by taking the average of experimental result. So it should be more general conclusions.

Experiments consist of N virtual servers and M requests. Requests is divided into 10 types (usually a service request type can be classified based on URL requests, such as news, fashion classes, music classes, etc.). Since for a heterogeneous network, it has been fixed to 10% to have a reasonably difficulty with the load balancing, therefore we have 10 different types of requests [8]. For simulating a real network, these types of requests and the link between them are randomly generated. In the experiment, all the requests are able to be processed in 5 ticks, and the case in which 70% of the requests require 7 ticks and 30% 3 ticks. The probability (px) that an unsevered request reads the advert board is initially set to 0.2, while the probability (pr) that a successful severed request writes to the advert board is 0.8. Finally, an advertisement’s lifespan on the advert board is equal to 10 ticks.

There are two phases of experiments in the following section and the strategy as described above. The first phase measures throughput against the number of requests; experiments run for \( N = 1000 \), and an increasing value of \( M \) (10 to 800) for each iteration. The second phase measures throughput against available servers; experiments run for \( M = 50 \), and an increasing values of \( N \) for each iteration is from 100 to 350.

Results and discussion:
Phase 1:

Fig. 1 shows, in certain number of servers, changes on the number of requests impact on system throughput, with the x axis showing the effect of increased requests on performance:

![Throughput vs. Quantity of Request](image)

**Figure 1. Throughput vs. Quantity of Request in a fixed size (1000 node) system**

Figure 1 demonstrates that when fixing the number of servers while increasing the number of requests, the algorithm for the original honeybee does not significantly improve in the system throughput, and it is always in a smaller range of fluctuation. In this case, although it curbs the reduction in system throughput, which dues to an increase in requests, it still can not meet practical needs. In the [6], although its system throughput better than the bees colony algorithm and improved algorithm in a less amount of the request, it will gradually decline as the amount of requests increase. It is particularly serious when the amount requested is increased from 10 to 100. Under the cloud computing environment, the algorithm which is not able to cope with a
considerable number of concurrent requests is not suitable for using. Improved algorithms for the honeybees, with the increase in the amount requested, its system throughput rises as logarithmic level. Compared to [6], in the small number of requests, the improved algorithm is not as well as [6]. It means a lower system throughput in the interval. However, when the request amounts to 800, the system throughput far exceeds the proposed method of [6]. So the improved algorithm is particularly suitable.

Phase 2:

Fig. 2 shows, in a certain number of requests, changes on the number of servers influence system throughput, with the x axis showing the effect of increased system size (resource availability) on performance:

![Throughput vs. Quantity of Sever](image)

This graph demonstrates that the original honeybee algorithm performs consistently again, and does not increase throughput in line with system size. Under the cloud computing environment, changing the number of servers is always happening. Therefore they need a relatively stable system. In this aspect, honeybees’ algorithm is suitable for cloud computing environments. However, the other approaches are able to utilize the increased system resources more effectively to increase throughput. Also, this is cloud computing needs, a better scalability. Under the premise of the appropriate initialization of forager bees, improving honeybee algorithm proposed in this paper is better than method in [6], and its performance is more prominent in scalability. So it needs to be combined honeybees algorithm. Under the circumstance that the number of servers increases or decreases, the load balancing algorithm should be done by honeybee in order to maintain system stability, and to prevent the system crash. After adjusting the server, it should use the improved honeybee algorithm to increase system throughput.

5. Conclusions and prospects

This paper presents a load balancing mechanism based on artificial bee colony algorithm. Through imitation of behavior of honey bees, it optimizes the amount of nectar (i.e., system throughput) to reach the maximum throughput. Since colony algorithm arranges only a little link between requests in the same server queue, the improvement of system throughput is suboptimal. Furthermore, in a certain number of servers, the increasing request does not lead to the increase of system throughput. Therefore, this paper proposes an improved bee colony algorithm to solve this problem. However, to a certain number of requests, the original algorithm shows better stability than others. Therefore, further study will focus on the combination of honeybees’ algorithm which has a better system stability and scalability.

6. References


