A Dynamic Load Balancing Strategy Based on Improved Ant Colony Algorithm

HangQi Shi¹, ZiLi Hao² and HangQi Shi*¹

¹Software engineering, Beijing University Of Technology, BeiJing, 100124, China
²Software engineering, Beijing University Of Technology, BeiJing, 100124, China
*Corresponding author’s e-mail: shq@emails.bjut.edu.cn

Abstract. Load balancing technique plays a fundamental role in distribution, while facing numerous challenges. It is difficult for ordinary load algorithms to cope with complex scenarios, and it is likely that uneven load will lead to downtime of a single server, thus causing an avalanche effect and eventually breaking down the whole cluster. This paper proposes a load balancing strategy based on improved ant colony algorithm, considers the influence of task length and processing efficiency in the algorithm based on the original ant colony algorithm, and takes the load rate of nodes as an important index for ants to select path. In addition, the load coefficient includes CPU, memory, IO, and network bandwidth into the calculation range, which makes the obtained server load condition more accurate. Finally, test data of different algorithms are obtained through experiments, and the advantages and disadvantages of this load balancing strategy are verified after horizontal comparison.

1. Introduction

With the rapid spread of 5G network in recent years, more and more equipment has started to access network, which has increased the load on the device access platform and has brought greater challenges to the server performance[1]. The processing capacity of a single server is far from meeting demands, while the emergence of distribution has solved the performance bottleneck of a single server. Load balancing is a key element in the existing distributed system[2]. Load balancing is the distribution of total load to each node in the cluster, and the core problem of load distribution is to provide the minimum request imbalance and the maximum resource utilization rate to the nodes in the cluster [3]. From the perspective of load strategy, load balancing is divided into IP-based, DNS-based, and scheduler-based load balancing strategies. In terms of load algorithm, it is divided into static and dynamic algorithms. Static algorithm requires transcendental knowledge of the system, but can not cope with a dynamic environment with real-time changes. Dynamic algorithm will rely on various states of the current system for task assignment. These algorithms all aim to make the system more stable and thus improve the performance of the whole cluster.

At present, simple load balancing algorithms can not cope with problems when facing various complex scenarios, so more and more load balancing algorithms have been proposed and applied. In the literature[4], a workflow-based load balancing scheduling algorithm was proposed in a server job scheduling application scenario to estimate the execution time of jobs on the server based on the server status, find the shortest total response time, and convert job scheduling problem into the minimum system response time problem. The literature [5] mentioned that in grid scheduling application scenarios, the use of traditional load balancing algorithm or traditional grid scheduling algorithm could not meet
the demand after the increase of complexity and task size, and that an optimized hybrid genetic algorithm was used to solve the multi-objective grid scheduling problem combined with its dynamic performance. The above research results show that the load algorithm has different optimization directions and algorithm options after combining specific complex scenarios.

This paper proposes a dynamic load algorithm based on the improved ant colony algorithm, which is a highly self-adaptive algorithm improved on the original ant colony algorithm. Unlike other algorithms that only use CPU utilization as the monitoring item, the more complex monitoring conditions are used as the judging criteria this time. Four conditions of CPU utilization, memory utilization, IO utilization and network bandwidth, which use different weights to represent the importance degree of such condition in the load rate, make the dynamic data obtained by the algorithm more accurate. Moreover, task length and load rate are also added to the load algorithm as influencing ant factors to enable the algorithm to converge faster. The performance of the algorithm is evaluated by simulation experiments, and the experimental results show that the algorithm is superior to other conventional algorithms in terms of load balancing capability.

2. Scheduling Model Design

2.1. Load Expression

Load information collection module is distributed in each base server, and load information of the server is obtained at intervals of \( \Delta t \) by means of timed tasks. In this paper, the CPU utilization, memory utilization, IO utilization and network bandwidth utilization are used to comprehensively reflect the load condition of the server. Here, \( (t_{k-1}, t_k) \) is defined as the time point of \( k \), and the load condition of the \( i \)-th server at the \( k \)-moment of the current time point is represented by various indexes of the time frame \( \Delta t \) before the \( k \)-moment.

\[
U(i, k) = w_c * \bar{U}_{cpu}(k) + w_m * \bar{U}_{mem}(k) + w_d * \bar{U}_{io}(k) + w_w * \bar{U}_{net}(k)
\]

Where

\[
\begin{align*}
\bar{U}_{cpu}(k) &= \frac{1}{2} \left( U_{cpu_{tk}} - U_{cpu_{tk-1}} \right) \\
\bar{U}_{mem}(k) &= \frac{1}{2} \left( U_{mem_{tk}} - U_{mem_{tk-1}} \right) \\
\bar{U}_{io}(k) &= \frac{1}{2} \left( U_{io_{tk}} - U_{io_{tk-1}} \right) \\
\bar{U}_{net}(k) &= \frac{1}{2} \left( U_{net_{tk}} - U_{net_{tk-1}} \right)
\end{align*}
\]

The symbol \( U_{cpu_{tk}} \) represents the CPU utilization of the server at the \( k \)-moment. The average of the CPU utilization at the \( k-1 \)-moment and \( k \)-moment is used to represent the approximate cpu utilization during the \( k \)-th time frame. Similarly, \( U_{mem_{tk}}, U_{io_{tk}} \) and \( U_{net_{tk}} \) also indicate similar meanings, while \( w_c, w_m, w_d \) and \( w_w \) are weighting coefficients of the four performance indexes in determining the load condition.

2.2. Load Balancing Evaluation Model

The first approach is to represent the load dispersion degree between each server though variance. The load expression \( U(k) \), as mentioned before, is also used in load balancing evaluation. Here the load variance of each server is adopted to make an evaluation of the load condition of the whole system.

\[
\sigma(k) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \left( \bar{U}(k) - U(i, k) \right)^2}
\]

Where
When the value of $\sigma(k)$ is larger, it means that the load of the whole system is more unbalanced and the overall resource utilization of each server in the system is worse. Therefore, when designing the load algorithm, the value of $\sigma(k)$ needs to be as small as possible to ensure that there is no overload on a server in the system.

The second approach is to represent the processing time of server through response time. Another index to judge the server load is the response time of request. The response time is the time consumed from the time the request is sent to the time when such request obtains the corresponding result from the server. The shorter the response time, the stronger the performance of server cluster and the better the performance of load algorithm.

3. Dynamic Ant Colony Algorithm Implementation

3.1. Ant Colony Algorithm

The ant colony algorithm [3], also known as the ant algorithm, was first proposed by Australian scholars Marce Dorigo et al. in 1992, which was inspired by the fact that ants could always find the shortest path between anthill and food in the process of looking for food. At the beginning, ants will randomly select some paths. When some ants find food, they will bring food back to the anthill, while leaving pheromones on the path, and other ants will select path according to the concentration of pheromones. As more and more ants find the shortest path, the concentration of pheromones will increase until all ants can find food.

This paper proposes an improved dynamic ant colony algorithm, which adds the influence of task length and processing efficiency on the algorithm based on the original ant algorithm, and also uses the load rate of nodes as an important index for ants to select path, so that ants will obtain the latest load rate of each server in each round of path selection as a reference weight for the current ants in path selection.

3.2. Algorithm Implementation Process

The roulette wheel method will be used by ants to select the next node in each step of path construction. One of the most important formulas involved is as follows.

$$
\frac{1}{N} \sum_{i=1}^{N} U(i,k)
$$

Where $\frac{1}{N} \sum_{i=1}^{N} U(i,k)$ represents the probability that the $k$th ant chooses to go from $i$ to $j$, and $\tau_{ij}$ is the concentration of pheromones from $i$ to $j$ at the $t$ moment. $\alpha$ is the pheromone factor, reflecting the relative importance degree of pheromone concentration in making choice by ants. $\eta_{ij}$ is the reciprocal of load coefficient from $i$ to $j$ at the $t$ moment, where $\beta$ is the heuristic function factor, reflecting the strength of the influence of priori and deterministic factors in determining the selection path in the searching process of ants. allowed $k$ is the set of nodes that have not been accessed yet. Here, assuming that the processing capacity of each server is the same, the task length shows a linear relationship with the load coefficient.

$$
U_{ij}(t) = \frac{\tau_{ij}^{\alpha}(t) \cdot \eta_{ij}^{\beta}(t)}{\sum_{j \in \text{allowed}_{k}} \tau_{ij}^{\alpha}(t) \cdot \eta_{ij}^{\beta}(t)}, j \in \text{allowed}_{k}
$$

$$
U_{ij}(t) = \frac{\tau_{ij}^{\alpha}(t) \cdot \eta_{ij}^{\beta}(t)}{\sum_{j \in \text{allowed}_{k}} \tau_{ij}^{\alpha}(t) \cdot \eta_{ij}^{\beta}(t)}, j \notin \text{allowed}_{k}
$$

Where $P_{ij}^k$ represents the probability that the $k$th ant chooses to go from $i$ to $j$, and $\tau_{ij}$ is the concentration of pheromones from $i$ to $j$ at the $t$ moment. $\alpha$ is the pheromone factor, reflecting the relative importance degree of pheromone concentration in making choice by ants.

$\eta_{ij}(t) = 1/U_{ij}$ is the reciprocal of load coefficient from $i$ to $j$ at the $t$ moment, where $\beta$ is the heuristic function factor, reflecting the strength of the influence of priori and deterministic factors in determining the selection path in the searching process of ants. allowed $k$ is the set of nodes that have not been accessed yet. Here, assuming that the processing capacity of each server is the same, the task length shows a linear relationship with the load coefficient. $U_{ij}(t)$ in the above equation is:

$$
U_{ij}(t) = U_{ij}(t-1) + \varphi L_{\text{task}}
$$

$\varphi$ is the performance coefficient and is a constant. $L_{\text{task}}$ is the length of the current task.

In addition, in every iteration, pheromone update will be carried out after all ants have selected path, and the pheromones on the path of that node will increase only when ants have selected that node. The concentration of pheromones at the $t+1$ moment is the sum of concentration of pheromones at the $t$
moment multiplied by the residual coefficient of pheromones and the amount of pheromones left by all ants on that path in this iteration. The specific formula is as follows:

$$\tau_{ij}(t + 1) = \tau_{ij}(t) \times (1 - p) + \Delta \tau_{ij}$$,

0 < p < 1

Where

$$\Delta \tau_{ij} = \sum_{k=1}^{m} \Delta \tau_{ij}^k$$

In the formula, ρ is the volatilization coefficient, which reflects the level of pheromone disappearance and takes a value range usually between 4.6 and 4.9. On the contrary, 1 - ρ reflects the level of pheromone retention, which is the residual coefficient of pheromones. If the value of pheromone volatile speed is too large, it will easily lead to elimination of a better path, while a small value of pheromone volatile speed will reduce the convergence rate, increase the number of iterations, and lead to the overall efficiency decline of algorithm.

3.3. Algorithm Calling Process

Because the algorithm needs a certain amount of requests to start, it requires to use a task queue to temporarily store all the requests and start executing the algorithm when a certain amount or a certain time is reached, and the load schedule module is responsible for controlling the whole algorithm calling process. When the load capacity of a server exceeds a preset threshold, the dynamic ant colony algorithm will be started. The requests will be temporarily stored in the task queue and the number of ants will be dynamically generated according to the number of tasks, so as to start executing the ant colony algorithm. See Algorithm 1.

**Algorithm 1**

If load condition of server exceeds the threshold
- Take M tasks from the current task queue and get the task length.
- While i<N-1 && isNotBalance do
  - Produce ants
  - Perform forward traversal and assign tasks based on the task load
  - Update the pheromone information
  - i++
- end while
- Find the optimal task assignment sequence based on the result set
- Assign tasks to each server based on the optimal sequence

4. Experiment And Test

In this chapter, a simulated experimental environment is built in order to test the practical effect of the algorithm. In the experiment, the convergence of the algorithm is tested under different concurrent quantities. In addition, the result of the algorithm in this paper is carried out horizontal comparison with other existing methods to prove the effectiveness of such method.

4.1. Environment Setup

The experimental environment is mainly set up by cloud servers, application containers, virtual machines, open source tools, etc. A total of five cloud servers are used as experimental machines, of which two of them are used as a client for test connection and a load module server, while the remaining three are used as base servers. On the base server, java is used as the development language, and TCP protocol is used for transmission of heartbeat detection and information interaction between load module and basic services.

A custom load module is used in the distributed architecture. This module is responsible for request processing, load monitoring, and load algorithm selection, where a pluggable design idea is used to switch the current load algorithm arbitrarily.
Various request combination modes are used for requests, allowing hybrid sending of long and short requests, so that the using effect of the algorithm in the complex environment can be tested.

4.2. Experimental Result

The results of Round Robin Algorithm (RRA), Least Connection Algorithm (LCA) and Dynamic Ant Colony Algorithm (DACA) proposed in this paper are compared in the experiment. In order to make the results more accurate, each connection is tested five times and the average value is taken as the final result. The obtained data are shown in Table 1 and Table 2.

| No. of Request | RRA/ms | LCA/ms | DACA/ms |
|----------------|--------|--------|---------|
| 200            | 35.2   | 33.1   | 43.9    |
| 400            | 40.6   | 37.4   | 50.3    |
| 600            | 42.4   | 45.3   | 54.4    |
| 800            | 135.4  | 80.6   | 104.6   |
| 1000           | 468.9  | 178.4  | 231.5   |
| 1200           | 632.5  | 353.2  | 310.4   |
| 1400           | 758.4  | 589.5  | 374.5   |
| 1600           | 1037.5 | 659.3  | 456.3   |
| 1800           | 1386.4 | 796.1  | 643.9   |
| 2000           | 1538.4 | 1047.6 | 834.2   |

| No. of Request | RRA   | LCA   | DACA  |
|----------------|-------|-------|-------|
| 200            | 1.21  | 1.11  | 0.96  |
| 400            | 1.35  | 1.04  | 1.04  |
| 600            | 1.44  | 1.27  | 1.07  |
| 800            | 1.28  | 1.21  | 1.04  |
| 1000           | 1.22  | 1.37  | 1.11  |
| 1200           | 1.31  | 1.42  | 1.04  |
| 1400           | 1.39  | 1.31  | 0.99  |
| 1600           | 1.18  | 1.36  | 1.06  |
| 1800           | 1.47  | 1.41  | 1.03  |
| 2000           | 1.43  | 1.37  | 1.05  |

It can be seen from the table that when the concurrent quantity is low, the difference among the concurrent quantity of the three algorithms is small, and sometimes the concurrent quantity of dynamic ant colony algorithm is lower than that of the other two algorithms, but when the concurrent quantity reaches more than 800, there is a significant improvement, and the response time of other algorithms increases rapidly. This is because when the concurrent quantity is low, the calculated quantity of dynamic ant colony algorithm is relatively large and the time delay is significant, while with the increase of concurrent quantity, the computing time of the algorithm decreases in proportion to the service execution time, so the response time will increase slowly.

From the data of load variance, it can be seen that the variance of both roll polling algorithm and least connection algorithm is large in different concurrent connections due to the different long and short tasks in the complex environment, while the load variance of dynamic ant colony algorithm is smaller than the other two algorithms in different connections, and the overall load variance has a small fluctuation, which indicates that the dynamic ant colony algorithm is superior to other algorithms in terms of load balancing.
5. Conclusions And Future Work

5.1. Conclusions
This paper proposes a load balancing strategy with high availability and stability. The algorithm is slightly less efficient than various static load algorithms and dynamic load algorithms under low concurrent quantity, but the load condition of each node in the cluster will be better. The advantage of this algorithm is more significant under high concurrent quantity, which not only greatly reduces the response time, but also has no large fluctuation of server load condition and still maintains a relatively stable status. In general, the algorithm has a better effect in the complex environment with high concurrent quantity.

5.2. Future Work
Combining the previous research on static and dynamic algorithms, it is found that the simpler and more direct algorithms have a higher efficiency in the actual using process, among which the roll polling algorithm has the highest efficiency. The dynamic ant colony algorithm proposed in this paper has no advantage in the low concurrent quantity because the load ondex of each server needs to be obtained dynamically during the algorithm, and the calculated quantity in the whole algorithm is also more complex.

In the future work, a hybrid algorithm mode will be attempted. When the load is low in each server, the static roll polling algorithm with high efficiency will be used. The algorithm is to roll pulling requests and assign them in turn in different servers theoretically. The advantage is that it has high execution efficiency and short request response time. However, the disadvantage is also obvious. When the request time is long, it may cause problems of unbalanced server load and overload on a single server. When the load on the server exceeds the set threshold or the task queue length is too long, the schedule module will switch the current load algorithm to use the dynamic ant colony algorithm for task processing, which can take advantage of different algorithms.

Reference
[1] Mogi, R., Nakayama, T., Asaka, T. (2018) Load Balancing Method for IoT Sensor System Using Multi-access Edge Computing. In: 2018 Sixth International Symposium on Computing and Networking Workshops (CANDARW). Takayama. pp. 75-78.
[2] Mbarek, F., Mosorov, V. (2018) Load balancing algorithms in heterogeneous web cluster. In: 2018 International Interdisciplinary PhD Workshop (IIPhDW). Swinoujście. pp. 205-208.
[3] Lei, Z., Xiang, J., Zhou, Z., Duan, F., Lei, Y. (2012) A multi-objective scheduling strategy based on MOGA in cloud computing environment. In: 2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems. Hangzhou. 2012. pp. 386-391.
[4] Li, C., Tang, J., Ma, T., Yang, X., Luo, Y. (2019) Load balance based workflow job scheduling algorithm in distributed cloud. Journal of Network and Computer Applications, 152, 102518.
[5] Ankita, Sahana, S. (2019) Evolutionary based hybrid ga for solving multi-objective grid scheduling problem. Microsystem Technologies, 26, 13–15.