Ant Colony Algorithm for Time Optimal Load Balancing Scheduling Problem Strategy

Yuxiang Tong¹, Xiaohui Li²*

¹ North China Institute of Computing Technology, Haidian, Beijing, 100000, China
² North China Institute of Computing Technology, Haidian, Beijing, 100000, China
*Corresponding author’s e-mail: 0902150113@csu.edu.cn

Abstract. In order to optimize the task execution efficiency of FPGA high-performance network clusters, this paper proposes a time-optimal load balancing scheduling problem algorithm and initialization strategy. Taking the task completion time as the optimization direction, the load balancing scheduling problem is abstracted as solving the placement matrix problem. On this basis, a node selection strategy using critical matrix is proposed, and the algorithm design is completed based on the ant colony algorithm. Aiming at the problem of low efficiency of random initialization, this paper proposes two ant colony initialization methods, which are ranked greedy initialization strategy and polling decentralized initialization strategy. Experiments show that the algorithm and initialization strategy designed in this paper can adapt to different computing scales, and the two initialization strategies can achieve a 15%~55% improvement in the iterative convergence times and load balancing compared to random initialization.

1. Introduction

The FPGA high-performance network cluster uses TOE[1](TCP Offload Engine) technology to receive high-speed network data streams, and directly completes the hardware offloading of the TCP/IP protocol stack on the FPGA. Compared with traditional servers, FPGA nodes using TOE technology can meet the requirements of high data throughput and low latency. An important issue that needs to be considered when combining multiple FPGA nodes into a cluster is load balancing scheduling. Load balancing scheduling studies resource allocation strategies within a certain period of time[2]. By optimizing the utilization of resources, it can achieve the optimal execution time, load balancing, throughput and other indicators. The problem of load balancing scheduling is also a very important research direction in modern distributed systems. A good load balancing strategy can maximize the operating efficiency of a multi-node system.

The load balancing studied in this paper is global static load balancing. From the perspective of classification, load balancing scheduling can start from local or global[3]. According to different distribution strategies, load balancing scheduling can be divided into static load balancing and dynamic load balancing[4]. The local load balancing scheduling studies the data exchange in a certain group, while the global load balancing will pay attention to the allocation of all resources in the system, which is more versatile. Static load balancing studies the task allocation under a given resource, while dynamic load balancing continuously adjusts the node load according to the real-time changes of the system. Dynamic load balancing considers the current operating state and needs to continuously collect system information, so additional resource consumption is often generated. Correspondingly, static load balancing has low overhead, fast execution speed, and high stability.
The global static load balancing problem is NP complete problem[5]. As a common combinatorial optimization algorithm, ant colony algorithm has great advantages in solving difficult NP discrete optimization problems. This article considers using ant colony algorithm to design optimization strategies for load balancing scheduling problems. At present, there are many designs and implementations that use heuristic algorithms to solve the load balancing scheduling problem at home and abroad. For example, literature [6] improves resource utilization by combining resource status and ant colony algorithm; literature [7] improves the problem of multi-objective constraints. Swarm algorithm minimizes task execution time; Literature [8, 9] proposed a particle swarm optimization algorithm considering task priority; Literature [10] considered improving resource utilization and task execution time, and proposed an improved PSO algorithm; Literature [10] proposed a virtual machine placement strategy, improved the ant colony algorithm, and compared and analyzed the first-in, first-out and polling algorithms; literature [11] proposed an improved firefly algorithm, taking into account the current network load situation and historical data.

It can be found that the above researches have improved the intelligent algorithms for different optimization directions, including load balance, resource utilization, and so on. This article focuses on FPGA high-performance network clusters and pays more attention to the time delay of task processing, hoping to process large batches of data within a certain period of time. At present, the research on optimal load balancing scheduling for a single time is not perfect, so this paper considers the use of minimizing task completion time to model the load balancing scheduling problem, and designing a load balancing scheduling scheme based on ant colony algorithm by optimizing the overall task completion time. In addition, this paper also proposes two initialization strategies to improve the execution efficiency of the algorithm.

2. Problem description and mathematical modeling

2.1. Problem Description
The load balancing scheduling problem studies the allocation of n problems of different scales to m computing nodes with different processing capabilities, and each node is processed in parallel, and the completion time of all tasks is used as an indicator to measure the excellent distribution strategy. Since each node is executed in parallel, the node computing time that takes the longest time can be used as the completion time of all tasks.

2.2. Model assumptions
1. The time consumption of each computing node during task switching is 0
2. In addition to processing tasks, there is no additional time and resource consumption for computing nodes
3. Each computing node completes the tasks assigned to that node at a constant processing speed
4. There is no mutual interference between computing nodes and tasks

2.3. Mathematical problem abstraction and model establishment
The time-optimal load balancing scheduling problem requires the design of a task allocation strategy to minimize the completion time of all tasks. To this end, we can set up a 0-1 placement matrix $C$ to represent the distribution of all tasks at each node.
Figure 1  Schematic diagram of placement matrix C

The first dimension of the matrix represents the task, and the second dimension represents the node. Suppose the matrix is $C$, $C_{ij} = 1$ means that task $i$ selects node $j$ as its processing node. As shown in Figure 1, we can add and calculate the processing time of each node according to the second dimension of the matrix, which is the dimension of the node, compare the processing time of each node, and finally use the maximum processing time as the task allocation strategy operation hours. Therefore, suppose the task set $T = \{T_1, T_2, \ldots, T_n\}$, which represents the scale of the task, and set the node set $N = \{N_1, N_2, \ldots, N_m\}$, which represents the processing capacity of the node. The final execution time of the entire strategy can be expressed as

$$E = \max \left\{ \sum_{i=1}^{n} C_{i1} \frac{T_i}{N_1}, \sum_{i=1}^{n} C_{i2} \frac{T_i}{N_2}, \ldots, \sum_{i=1}^{n} C_{im} \frac{T_i}{N_m} \right\}$$  (1)

Among them, $\frac{T_i}{N_j}$ represents the running time of task $i$ on node $j$, multiplied by the selection matrix $C$ and summed to get the running time of node $j$. Since each node is executed in parallel, the maximum running time of all nodes is the overall operation of the strategy time. The entire load balancing scheduling problem is simplified to finding the placement matrix $C$ so that the value of the execution time $E$ reaches the minimum.

3. Load balancing scheduling algorithm based on ant colony algorithm

In this section, the algorithm design and implementation are carried out for the mathematical modeling of the load balancing scheduling problem of the placement matrix proposed in the second section, and the ant colony algorithm is used to search for the optimal placement matrix. For the load balancing and scheduling problem, the traditional ant colony algorithm is improved accordingly, and the concept of critical matrix is proposed to facilitate the optimal selection of nodes. Finally, a whole set of algorithm schemes for the processing flow of load balancing and scheduling problems are proposed.

3.1. Node selection strategy using critical matrix

In the ant colony algorithm, the path with the largest pheromone represents the current optimal solution, but all ants cannot be allocated to the path with the largest pheromone, otherwise it will fall into a local optimal solution. Here we judge whether to choose the path with the largest pheromone based on the number of the ant. Suppose the critical matrix $criticalM$ is a one-dimensional matrix, and the size of the matrix is equal to the number of tasks. The value of the matrix is expressed as

$$criticalM_i = \left[ antNum \times \frac{\text{maxPheromone}}{\sum_{j=1}^{n} \text{Pheromone}_{Mij}} \right]$$  (2)

When selecting the task processing node for the current ant, if the ant number is less than the value of the critical matrix $criticalM$, the node with the largest current pheromone is selected, otherwise the node is randomly selected. $\text{maxPheromone}$ is the maximum pheromone, and $antNum$ is the number of ants. It is worth mentioning that in the process of algorithm implementation, the node corresponding to the $\text{maxPheromone}$ value needs to be recorded. If all the pheromone sizes are the same at this time, set the recorded node as a random node.
The critical matrix controls the number of ants that randomly choose a path, which is conducive to finding better solutions. At the same time, the ratio of the maximum pheromone value to the total pheromone value can indicate the certainty of the optimal solution. Using the maximum pheromone size to control the value of the critical matrix can also accelerate the convergence of the optimal value.

3.2. Pheromone Update Strategy

Use the value $P$ to control the pheromone attenuation ratio, $P$ is the volatilization factor, $P \in (0, 1)$, simulating the pheromone secreted by the ants in the real ant colony and gradually attenuates over time. After one iteration is completed, all values of the pheromone matrix are multiplied by the value $P$.

Use the value $Q$ to control the ratio of pheromone strengthening, $Q$ is the strengthening factor, $Q \in (1, +\infty)$, which simulates the more path pheromone the ants pass through in the real ant colony. After an iteration is completed, find the ant number with the smallest overall execution time $E$, and multiply the pheromone value corresponding to all nodes selected by the ant by the value $Q$.

3.3. Overall realization process

The overall implementation process of the ant colony algorithm in the load balancing scheduling problem is shown in Figure 2 pseudo code, where $P_{all}$ represents the path placement matrix of all ants, and $P_{one}$ represents the current ant path placement matrix. First, iterate according to the maximum number of iterations. Each ant must find the placement matrix for all tasks. When assigning a task, obtain the selected node number according to the node selection strategy using the critical matrix proposed in Section 3.1. After an iteration is completed, update the pheromone matrix and other information according to the pheromone update strategy in section 3.2.

```
Algorithm acaSearch
1. for each iteration number
2. $P_{all} \leftarrow Null$ // Record all node selections of all ants
3. for each ant number
4. Initialize $P_{one}$ to all zeros; // Record all node selections of current ant
5. for each task number do
6. Assign $P_{one}$ value based on critical matrix selection strategy;
8. end for
9. $P_{all} \leftarrow P_{all} \cup P_{one}$;
10. end for
11. Calculate the running time of all ants in this iteration;
12. Use Pheromone Update Strategy to update;
13. end for
```

Figure 2 The overall flow of the algorithm
4. Initialization strategy

Generally, ant colony algorithm often uses random initialization when initializing paths. Although random initialization can generate multiple paths, which is conducive to finding the optimal solution, if the value of random initialization is poor, it will often lead to an increase in the number of iterations, or it is difficult to find the best solution. Excellent solution. This paper proposes two initialization schemes, one is the ranked greedy initialization strategy to make the initialization result closer to the optimal solution, and the other is the polling decentralized initialization strategy to disperse the ant colony as much as possible.

4.1. Ranked Greedy Initialization Strategy

For a task, selecting the node with the strongest computing power can minimize the running time of the task, but from the overall point of view, it is more important to assign larger tasks to nodes with strong computing power. This trend can also be found by analyzing the placement matrix after the iteration, that is, larger tasks are often assigned to computing nodes with strong computing capabilities. Therefore, we can design an initialization strategy to make the initialization result closer to the normal operation result and optimize the iterative process of ant colony.

In this regard, this paper designs a ranking greedy initialization strategy, which allocates larger tasks to computing nodes in order. The steps are as follows:

1. Sort the computing nodes from largest to smallest in terms of computing power.
2. Sort the computing nodes according to the task length from largest to smallest, that is, get the size order of the length of each task among all task lengths.
3. Initialize half of the ants according to the following method: set the initial node number of the task to \( \text{taskSeq}[i] \% \text{nodeNum} \), where \( \text{taskSeq} \) is the numbered sorting matrix obtained in step 2, and \( \text{nodeNum} \) is the number of nodes. The other half of the ants keep initializing randomly.

4.2. Polling decentralized initialization strategy

In the load balancing scheduling problem, an important evaluation index is the degree of load balancing:

\[
l = \frac{1}{N} \sum_{i=1}^{N} (x_i - q)^2, \quad q = \frac{1}{N} \sum_{i=1}^{N} x_i
\]

Where \( x_i \) represents the number of tasks assigned by node \( i \), the load balance formula is to find the mean square deviation of the number of tasks assigned by the node. The calculation results of the formula can see whether the tasks are uniformly distributed to each calculation node. Low load balance value indicates that there is no significant difference in the number of tasks assigned to nodes.

Round Robin [12] method is a commonly used load balancing algorithm, which can maintain the load balancing degree at a low value. The polling method assigns tasks to computing nodes in turn, starting from the first node until the last node, and then repeats the cycle. Based on the idea of the polling method, this paper designs a polling decentralized initialization strategy to distribute tasks to each node. The steps are as follows:

1. Generate a random array matrix with a size equal to the number of tasks
2. Initialize half of the ants according to the following method: set the initial node number of the task to \( \text{randSeq}[i] \% \text{nodeNum} \) where \( \text{randSeq} \) is the random array matrix obtained in step 1, and \( \text{nodeNum} \) is the number of nodes. The other half of the ants keep initializing randomly.

5. Experiment analysis

5.1. Experimental setup

The operating system of this experiment is Ubuntu 18.04, and the algorithm is implemented using Matlab software. In the experiment, the P value is set to 0.5, the Q value is set to 2, and the number of ants is 101. The length of the task and the computing power of the node use random values, ranging from 10 to 100, and the same random value is used in each comparison test. The experiment explored
the impact of the three initialization algorithms under different task numbers and node numbers. The main research indicators are the iterative convergence algebra of the algorithm, the execution time of the algorithm, and the load balance value.

The number of iterations convergence reflects the operational efficiency of the algorithm, the lower number of iterations convergence explain the higher efficiency of the algorithm. A lower number of iterations can make the algorithm find the optimal solution faster, thereby improving production efficiency. Algorithm execution time is the time to complete all tasks, reflecting the execution speed of the algorithm. The load balance value can reflect whether the tasks are evenly distributed on the computing nodes. In a production environment, considering the lifetime of nodes, sometimes nodes with strong computing power are not expected to take on too many tasks, so it is hoped that the load balance value of the algorithm is as low as possible.

5.2. Experimental results and analysis

First, compare the load balancing algorithm designed in this article with the current mainstream load balancing algorithms, including genetic algorithm and Min-Max algorithm. Figure 3 shows the comparison of the three algorithms in task completion time. It can be found that the load balancing algorithm based on ant colony algorithm designed in this paper is better than genetic algorithm and Min-Max algorithm in task completion time. It proves that the algorithm in this paper has stronger performance in execution efficiency.

Next, compare the initialization strategy designed in this paper. The experiment sets up five different combinations of tasks and the number of nodes to explore the iterative convergence algebra, execution time, and load balance changes of different algorithms under different combinations. The task length and node computing power are randomly generated. In order to eliminate the influence of the randomly generated results on the algorithm, each test group will run the algorithm 5 times, so each data is the average of the results of 5 runs.

![Figure 3 Comparison of load balancing algorithms](image-url)
Figure 4 Comparison of iterative convergence algebra

Compare the iterative convergence algebra of the three algorithms, as shown in Figure 4. The abscissa is five different test groups, for example, 100, 10 means that the number of tasks in the test group is 100, and the number of nodes is 10, and the ordinate represents how many iterations the algorithm has reached convergence. It can be seen that the ranked greedy initialization strategy and the polling decentralized initialization strategy are better than random initialization in the convergence algebra, and as the number of tasks and nodes increase, the advantages become more obvious. Compared with random initialization, the two initialization algorithms improve by about 15% to 55%. In addition, ranking greedy initialization has a great advantage when the number of tasks and nodes is large. It is worth noting that in the 500 and 20 test groups, the iterative convergence algebra of the random initialization algorithm is relatively high, indicating that the random initialization is more dependent on the result of the initialization, and the execution efficiency of the algorithm will be greatly reduced in the case of poor initialization.

Figure 5 Algorithm execution time comparison

Figure 5 shows the comparison of the execution time of the algorithms. It can be found that the difference in execution time of the three algorithms is not obvious, and the ranked greedy initialization strategy has a slight advantage when the number of tasks and nodes increases. Compared with random
initialization, rank greedy initialization increases the task length and node computing power of the sorting process. The test results show that the two sorts do not increase the time cost of the algorithm.

![Load balance comparison](image)

**Figure 6** Comparison of algorithm load balancing degree values

Figure 6 shows the comparison of the load balance values of the three algorithms. Comparing the test results, it can be found that no algorithm can maintain its advantage when the number of tasks and nodes is small. In the group with a large number of tasks and nodes, the ranked greedy initialization strategy has achieved certain advantages, and the polling decentralized initialization is sub-optimal.

In general, the two improved algorithms have achieved a certain degree of improvement compared to random initialization. Among them, the ranked greedy initialization algorithm has greater advantages under the convergence algebra and load balance indicators, and the algorithm execution time is also slightly improved. Therefore, the ranking greedy initialization algorithm is worth popularizing in the actual production environment.

6. Concluding remarks

In this paper, mathematical modeling of load balancing scheduling problem with minimum completion time is carried out, the mathematical model of placement matrix is abstracted, and the mathematical model is analyzed using ant colony algorithm. Meanwhile, a node selection strategy using critical matrix is proposed. On this basis, this paper proposes two initialization schemes of ant colony paths, which are ranked greedy initialization strategy and polling scattered initialization strategy.

Intelligent algorithms represented by ant colony algorithm have great advantages in solving NP complete problems. Using ant colony algorithm based on critical matrix can improve the execution efficiency of load balancing scheduling problems. At the same time, using the improved initialization algorithm in this paper can further reduce the number of iterative convergence of the algorithm and improve the load balancing degree of the algorithm.

There are two directions worth considering in the next step. First, the experiment assumes that each task is independent of each other and has no priority. The algorithm can be redesigned by adding the task execution order and priority to adapt to the actual production environment. Secondly, there is still room for improvement in the polling decentralized initialization strategy. The algorithm can be optimized to further reduce the load balance value, and it can be extended to the situation with strong load balance requirements.

References

[1] HONG R. Research on the development and general situation of TOE technology [J]. Infrared, 2005, (3): 19-26.
[2] NASIRI, MAHDI M. A modified ABC algorithm for the stage shop scheduling problem [J]. Applied Soft Computing, 2015, 28: 81-9.

[3] ZAKI M J, WEI L, PARTHASARATHY S. Customized dynamic load balancing for a network of workstations [J]. Proc Hdp, 1997, 43(2): 156-62.

[4] QIAN X, YANG Q. An Analytical Model for Load Balancing on Symmetrical Multiprocessor Systems [J]. Journal of Parallel & Distributed Computing, 1994, 20(2): 198-211.

[5] ULLMAN J D. NP-complete scheduling problems [J]. Journal of Computer and System ences, 1975, 10(3): 384-93.

[6] LI K, ZHAO G, DAN W. Cloud Task Scheduling Based on Load Balancing Ant Colony Optimization; proceedings of the Chinagrid Conference (ChinaGrid), 2011 Sixth Annual, F, 2011 [C].

[7] DHINESH B, KRISHNA P V. Honey bee behavior inspired load balancing of tasks in cloud computing environments [J]. Applied Soft Computing Journal, 2013, 13(5): 2292-303.

[8] ENGIN O, CERAN G, YILMAZ M K. An efficient genetic algorithm for hybrid flow shop scheduling with multiprocessor task problems [J]. Elsevier Science Publishers B V, 2011, 11(3): 3056-65.

[9] KESHANCHI, BAHMAN, NAVIMIPOUR, et al. An improved genetic algorithm for task scheduling in the cloud environments using the priority queues: Formal verification, simulation, and statistical testing [J]. The Journal of Systems and Software, 2017, 124: 1-21.

[10] TAWFEEK M A, EL-SISI A, KESHK A E, et al. Cloud Task Scheduling Based on Ant Colony Optimization; proceedings of the Computer Engineering & Systems (ICCES), 2013 8th International Conference on, F, 2013 [C].

[11] XIQIAN Z, FENGLIAN L, XUEYING Z, et al. Multiple decision tree algorithm based on cost-sensitive hybrid splitting strategy [J]. Electronic technology application, 2017, (10): 134-7+42.

[12] ZILI Z. Network topology design and Hadoop platform research in cloud computing [D]; University of Science and Technology of China.