Proceedings of the 35th Chinese Control Conference
July 27-29, 2016, Chengdu, China

Load Balancing Task Scheduling based on Multi-Population Genetic Algorithm in Cloud Computing

WANG Bei, LI Jun

Department of Automation, University of Science and Technology of China, Hefei 230026, China

E-mail: wb5288@mail.ustc.edu.cn

E-mail: Ljun@ustc.edu.cn

Abstract: In this paper, a Multi-Population Genetic Algorithm (MPGA) considering load balancing is adopted for solving task scheduling problems in cloud environment instead of Genetic Algorithm to avoid premature convergence. In order to boost the search efficiency, the min-min and max-min algorithm are used for the population initialization. Moreover, Metropolis criterion is used in this paper to screen the offspring so that poor individuals can also be accepted with a certain probability, then the population diversity can be maintained and the local optimum can also be avoided. The simulation results show that a better task scheduling result (shorter completion time, lower processing costs, load balancing) could be achieved through the MPGA-based task scheduling algorithm, which means the algorithm can realize an effective task scheduling and is more suitable for handling quantities of tasks compared to adaptive genetic algorithm (AGA).

Key words: Cloud computing; Multi-Population Genetic Algorithm (MPGA); Load balancing; Task scheduling

1 Introduction

As a consequence of the development of parallel computing, grid computing and distributed computing techniques [1], cloud computing can provide users computing resources with an on-demand and pay-as-you-go way. It virtualizes underlying implements and avoids the direct interaction between users and cloud service providers. Only paying little, users can gain a strong capability of calculation without purchasing additional hardware. These advantages of cloud computing lead to its wide application and the growing number of customers. In order to meet the user needs, making full use of resources to obtain an efficient and reasonable task scheduling is very necessary.

There are many classical task scheduling algorithms, such as round robin algorithm, minimum link algorithm and min-min algorithm. They may perform well sometimes, but the shortcomings are also non-ignorable. Round robin algorithm and minimum link algorithm are not suitable for the situation where virtual machines have different performance. And min-min algorithm is likely to cause the problem of load imbalance. Many meaningful research works have been done to overcome these disadvantages. LI Jian-Feng, et al. proposed an improved-GA based double fitness task scheduling algorithm for less total and average task completion time in [2]. ZHENG Li presented an improved min-min algorithm for solving load balancing problems in [3] and WANG Ting-Ting, et al. improved the algorithm for less completion time and more load balancing [4]. Above all, these works either only considered the completion time or neglected the premature convergence of genetic algorithm.

Based on the discussion above, this paper aims to fulfill a task scheduling which takes into account the task execution time and cost as well as load balancing. Rest of this paper is organized as follows: in Section II, we briefly introduce the issue of task scheduling in cloud computing. Then the implementation details of the MPGA-based load balancing task scheduling algorithm are presented in Section III, followed by the experimental results and analysis in Section IV. Finally, the paper conclusion is given in Section V.

2 Task Scheduling Problem Description

In view of the high efficiency and versatility of the Map/Reduce programming model of Google [5], this paper will discusses the task scheduling problem in cloud computing based on this model. In cloud environment, once the user submits a task request, the task will be divided into many smaller subtasks in Map stage and be executed by multiple computing nodes in parallel. Then in Reduce stage, the intermediate results from Map stage will be summarized and analyzed. Consequently, the final processing result can be obtained and returned to the user.

Assume that there are *N* subtasks and *M* available computing nodes. Due to the different computing power among these nodes, the execution time of subtasks on each node is different. Besides, the processing cost of subtasks varies from one node to another. Suppose that the execution time of subtask on different nodes and the price of each node per unit time are known and can be expressed by matrix ETC and RCU respectively [5]. *ETC*(*i*,*j*) denotes the execution time of task *j* on node *i* while *RCU*(*i*) denotes the price of node *i* per unit time. Here, for easy explanation, we omit the units of time and costs, which means only focusing on the values. Based on the above introduction, the task scheduling problem is: assign *N* subtasks to *M* computing nodes reasonably as well as balance the load. In brief, task scheduling is seeking the mapping between tasks and virtual machines which meets the constraints and optimization goals.

3 Load Balancing Task Scheduling based on Multi-Population Genetic Algorithm

Genetic algorithm is a stochastic global searching and optimizing method inspired by Darwin's evolution theory [6]. It searches for the optimal solution by simulating natural evolutionary process. As one of the most extensive and efficient modern intelligent algorithm, genetic algorithm can get ideal results for a number of non-linear and multi-objective function optimization problems. Genetic Algorithms can be easily understood but are prone to premature convergence. Considering this, MPGA is used in this paper to solve the task scheduling problem in cloud system. MPGA strikes the optimal solution by parallel search of multiple populations [7]. As an improvement to standard genetic algorithm, it takes into account the balance between global and local search ability. Consequently, MPGA is able to achieve global optimization while avoid the premature convergence efficiently. Detailed descriptions about MPGA are given as below.

3.1 Chromosome Encoding

In this paper, a direct encoding scheme is adopted. The value of each gene in a chromosome denotes the corresponding node ID assigned to a subtask. Therefore, a chromosome represents a scheduling scheme. According to the previous assumptions, a chromosome has *N* gene bits. That is to say, the length of chromosome represents the number of subtasks and the value (1~*M*) of each gene represents the available computing nodes. For instance, there are 10 tasks and 5 nodes, then the length of chromosome is 10 and the value of each gene can be 1, 2, 3, 4, 5. Given a chromosome encoded as:

$$\{1,2,3,5,4,2,3,1,2,4\}$$

It denotes that task 1 will be executed on node 1, task 2 will be executed on node 2 …... and task 10 will be executed on node 4. The correspondence between computing nodes and tasks is illustrated in Table 1.

Table1: Correspondence between nodes and tasks

Nodes	Corresponding tasks
1	1, 8
2	2, 6, 9
3	3, 7
4	5, 10
5	4

Based on chromosome encoding, the matrix CONV whose element can only be 0 or 1 is constructed. If task *j* is assigned to node *i*, then *CONV*(*i*,*j*)=1, or *CONV*(*i*,*j*)=0. Therefore, each column has only one element for 1, with the rest all 0. For the example above, the corresponding CONV is

$$\mathrm{CONV} = \begin{pmatrix} 1\,0\,0\,0\,0\,0\,0\,1\,0\,0 \\ 0\,1\,0\,0\,0\,1\,0\,0\,1\,0 \\ 0\,0\,1\,0\,0\,0\,1\,0\,0\,0 \\ 0\,0\,0\,0\,1\,0\,0\,0\,0\,1 \\ 0\,0\,0\,1\,0\,0\,0\,0\,0\,0 \end{pmatrix}$$

According to matrix ETC, RCU and CONV, the running time of node *i* can be derived from Formula (1):

$$sumTime(i) = \sum_{j=1}^{N} ETC(i,j) \times CONV(i,j) \qquad (1)$$

All nodes work in parallel and the total execution time of all tasks is expected to be:

$$totalTime = \max_i\{sumTime(i)\} \qquad (2)$$

The execution cost of all tasks is

$$cost = \sum_{i=1}^{M} sumTime(i) \times RCU(i) \qquad (3)$$

In order to satisfy load balancing among nodes, the load imbalance value *load* is set as below:

$$load = \sqrt{\frac{1}{M-1}\sum_{i=1}^{M}(sumTime(i) - avgTime)^2} \quad (4)$$

Where *avgTime* is the average running time of nodes. It is visible that the value of *load* is smaller, the scheduling is more balancing and reasonable, and the resource utilization is higher [8]. In this paper, a threshold is set for the load imbalance value, which is used to constrain the evolution of the individual.

3.2 Initialization of Population

To reduce the complexity of MPGA, min-min and max-min algorithm are used to initialize part of the population. In this way, both the individuals scheduling small tasks firstly and the ones scheduling large tasks firstly can be found in the initial population. In order to maintain diversity of the population, random initialization method is still adopted for most part of the population. In this paper, the proportion between two parts is set to 1:3.

3.3 Fitness Function

In the genetic algorithm, the selection of fitness function directly affects the convergence rate of the algorithm and the quality of optimal solution. In order to balance time and cost, the fitness function should include two items at the same time.

The fitness function of time is

$$F_t = \frac{1}{totalTime} \quad (5)$$

The corresponding item of cost is

$$F_c = \frac{1}{cost} \quad (6)$$

The final fitness function can be obtained by combining the time and cost fitness function together:

$$F = \omega t \times F_t + (1 - \omega t) \times F_c \quad (7)$$

Where ωt denotes the weight of time in final fitness value and the corresponding one for cost is $(1 - \omega t)$, $\omega t \in [0,1]$. If we want to obtain a task scheduling scheme with minimum task execution time, the ωt can be set to 1. Similarly, setting ωt to 0 and then a task scheduling scheme with the lowest executing cost can be acquired. In practice, the value of ωt can be adjusted according to the bias for time and cost from users.

3.4 Genetic Operations

Standard genetic algorithm includes selection, crossover and mutation operations. As an improved GA, MPGA introduced immigration and elite reservation operations. Additionally, this paper adopted Metropolis criterion in MPGA to screen the offspring generated by genetic operations.

3.4.1 Selection Operation

In this paper, a deterministic sampling selection operator is used. The expected number of the i^{th} individual existing in next generation can be quantified as N_i, which is given by Formula (8).

$$N_i = NIND \cdot \frac{F_i}{\sum F_i} \quad (8)$$

Where *NIND* is the scale of each population and F_i stands for the fitness of the i^{th} individual. Therefore, $\sum_{i=1}^{NIND}[N_i]$ individuals can be determined in the next generation. Then we sort individuals in descending order according to the fraction part of N_i and add the first $NIND - \sum_{i=1}^{NIND}[N_i]$ individuals to the next generation. By far, *NIND* individuals have been determined.

3.4.2 Crossover and Mutation Operations

In order to improve the search efficiency, adaptive crossover and mutation operations are adopted in the paper. The probability of crossover is set as follows:

$$P_c = \begin{cases} k1 * \frac{f_{max} - f'}{f_{max} - f_{avg}}, & f' \geq f_{avg} \\ k2 \quad , & f' < f_{avg} \end{cases} \quad (9)$$

Where f' is the higher fitness of the two crossed chromosomes, and f_{max} is the maximum fitness of the population. The probability of mutation is set as:

$$P_m = \begin{cases} k3 * \frac{f_{max} - f}{f_{max} - f_{avg}}, & f \geq f_{avg} \\ k4 \quad , & f < f_{avg} \end{cases} \quad (10)$$

Where f is the fitness of the mutated individual. For the offspring generated by crossover and mutation operations, Metropolis criterion is used for screening. If the criterion is satisfied, the individual will be accepted and then be used to replace the corresponding parent. Otherwise, there is no replacement. Thus, it is even possible for poor individual to stay in next generation [9]. That is to say, the use of Metropolis criterion helps to maintain the diversity of the population, so as to avoid falling into the local optimum.

3.4.3 Immigration and Elite Reservation Operations

The immigration operation in MPGA adds the optimal individuals appeared in the evolutionary process into other populations regularly. It realizes the information exchange among populations. In elite reservation operation, the optimal individual appearing in each population will be picked out to constitute the quintessence population.

In this paper, the value of load imbalance is used as the constraint of an individual. In the process of evolution, only when the value is less than the set threshold [10] can individual be retained to next generation.

4 Results and Analysis

In order to verify the feasibility of the algorithm adopted in this paper for task scheduling in cloud environment, several experiments are conducted. And we simulate the cloud environment in the MATLAB.

4.1 Comparative Experiment between MPGA and AGA

In this section, comparisons between MPGA and AGA are given. And related parameter settings are shown in Table 2, where *MP* is the number of populations. The Matrix ETC and RCU are generated randomly.

If no significant change for the fitness of the optimal individual in the quintessence population happens for 10 generations, the iteration ends. Meanwhile, if the algorithm has been iterated for 150 generations, it will be forced to terminate as well.

Table 2: Parameter settings of the experiment

Algorithm	Parameters	Value
AGA	*NIND*	40
	k1	0.6
	k2	0.8
	k3	0.1
	k4	0.05
	ωt	0.25
	M	5
	N	50
MPGA	*MP*	10
	NIND	40
	k1	0.6
	k2	0.8
	k3	0.1
	k4	0.05
	ωt	0.25
	M	5
	N	50

Fig. 1 shows the comparison between AGA and MPGA adopted in this paper. The evolution process of MPGA is illustrated in Fig. 2. Here, the dashed line represents the evolution process of MPGA and the solid line represents the evolution process of AGA.

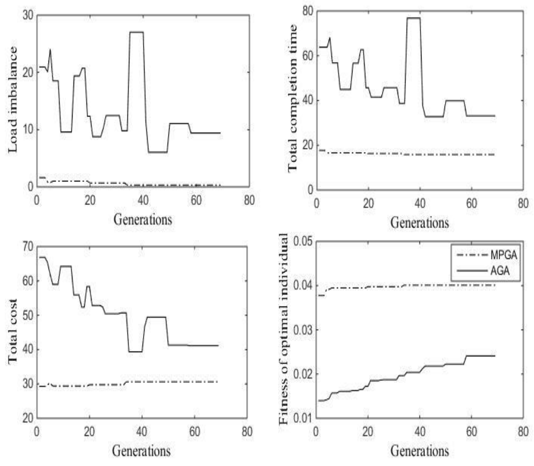

Fig. 1 The comparison between MPGA and AGA

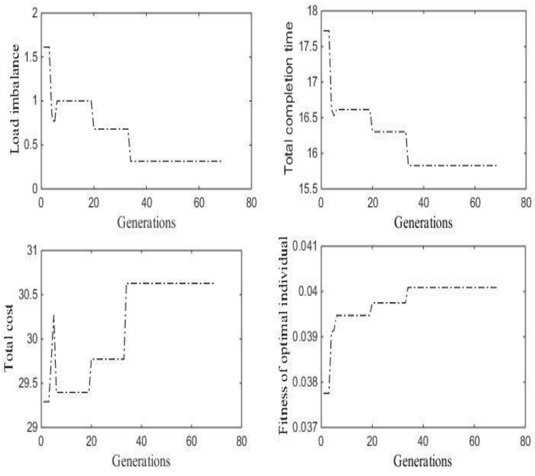

Fig. 2 The evolution process of MPGA

The optimal results of MPGA and AGA are presented in Table 3. From Table 3, we can know that MPGA is better than AGA in terms of total execution time and cost. Besides, MPGA has better load balancing performance.

Table 3: The optimal results of the experiment

Algorithm	*load*	*time*	*cost*
AGA	9.4121	33.1538	41.1537
MPGA	0.3162	15.8299	30.6286

As shown in Fig. 1, the optimal solution is obtained around 35^{th} generation for MPGA and after about 60 generations, AGA acquires its optimal solution. That is to say, MPGA can search the optimal solution faster than AGA. The better performance compared to AGA can be attributed to the initial population operation in MPGA, who use min-min and max-min algorithm for population initialization. Moreover, it can be seen from Fig. 1 that MPGA shortens the search time of the optimal solution efficiently. Besides, it is obvious that AGA finally falls into local optimum along with the evolution of the population for the reason that AGA always reserves the best individual. While the Metropolis criterion adopted in this paper makes it possible for poor individual to remain in next generation. At the same time, multiple populations searching in parallel also reduced the possibility of convergence to local optimum.

The performance comparison of MPGA-based and AGA-based task scheduling algorithms in terms of time and processing costs are shown in Fig. 3 and Fig. 4.

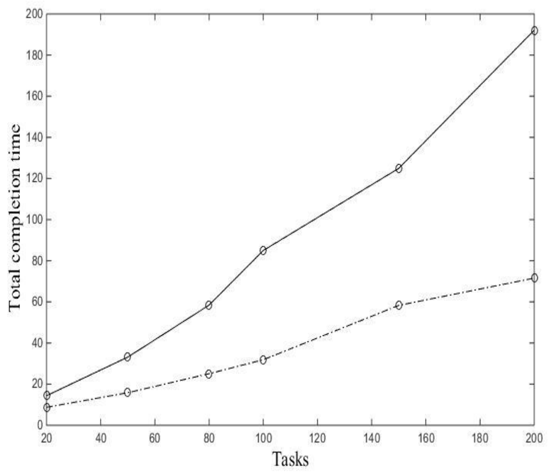

Fig. 3 Time varying with the number of tasks

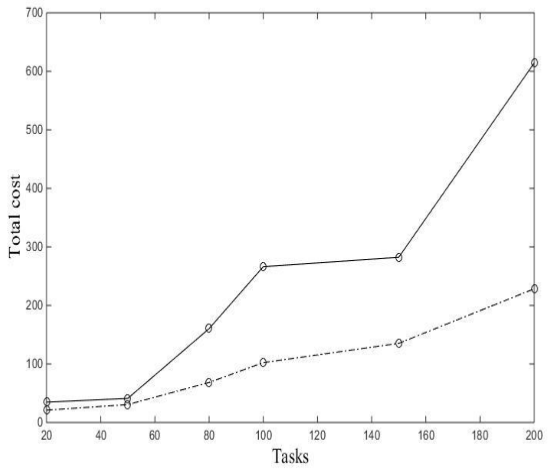

Fig. 4 Cost varying with the number of tasks

It is obvious that MPGA has better time and cost performance than AGA. Given the number of virtual machines, the advantage is more obvious with the increased number of tasks. The more tasks, the less the blindness of population evolution in MPGA. All of these advantages can be owed to the adoption of min-min and max-min algorithm. Generally speaking, the algorithm adopted in this paper is more suitable for large-scale task scheduling in cloud environment than AGA.

4.2 Performance Evaluation Experiment

Compared with TCGA (Time and Cost constraints Genetic Algorithm) in [5] and SAGA (Simulated Annealing Genetic Algorithm) in [9], the performance of MPGA can be further evaluated. In this section, ωt is set to 0.3. And other experiment parameter settings are the same as in Table 2.

The performances of three algorithms when M=5, N=80 are shown as Fig. 5. It is clear that MPGA performs best. Since the inter-nodes load balancing is neglected, the *load* of SAGA and TCGA undulate greatly while MPGA maintains a smaller load imbalance value.

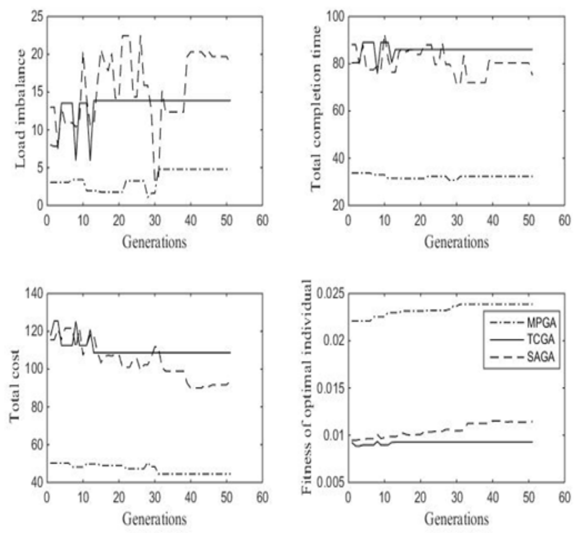

Fig. 5 The performance comparison when M=5, N=80

SAGA can partly solve premature of genetic algorithm, but it is still difficult for SAGA to gain the global optimal. As to TCGA, it takes time and cost as constraints but traps into local optimal.

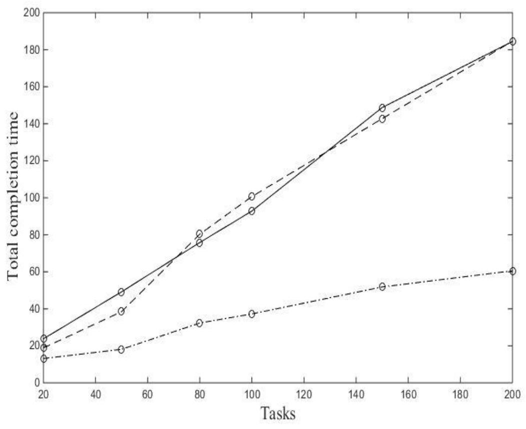

Fig. 6 Time varying with the number of tasks

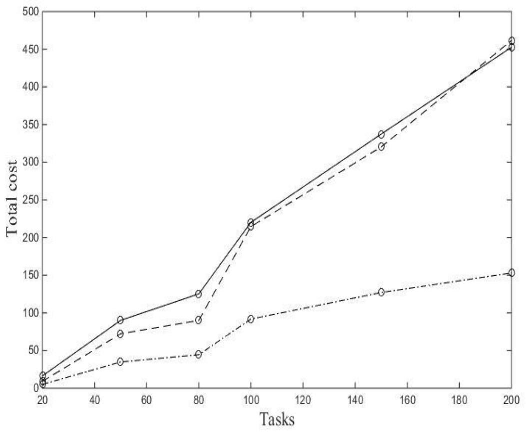

Fig. 7 Cost varying with the number of tasks

From Fig. 6 and Fig. 7, it is found that with the tasks number increasing, the performance differences between MPGA and the other two become distinct. SAGA is equal or better than TCGA in time and cost performance. For comparison, MPGA has a slower growth in time and cost even when the number of tasks increases a lot. From the experiment, it has been further proved that MPGA adopted in this paper can solve the problem of premature convergence well.

5 Conclusions

In this paper, a MPGA-based load balancing task scheduling strategy is presented and its good scheduling performance is testified by the simulation experiments. The MPGA-based scheduling strategy adopts min-min and max-min algorithm to initialize part of the populations and then use the Metropolis criterion to avoid local optimum. The experiment results show that the MPGA-based task scheduling algorithm has better performance than TCGA and SAGA in processing costs as well as time consuming, and it balances the inter-nodes load well. Besides, it can handle numerous tasks more effectively than AGA in cloud computing. Considering the coexistence of independent and related tasks in practical cloud environment, related tasks scheduling and dynamic scheduling should be taken into account in future research.

References

[1] N. Mohan and E. Raj, Resource Allocation Techniques in Cloud Computing--Research Challenges for Application, in *Proceedings of 4th International Conference on Computational Intelligence and Communication Networks*, 2012: 556-560.

[2] J. Li and J. Peng, Task Scheduling algorithm based on improved genetic algorithm in cloud computing environment, *Journal of Computer Application*, 31(01): 184-186, 2011.

[3] L. Zheng, The Research of resource scheduling key technology in cloud computing, Beijing: *Beijing University of Posts and Telecommunication*, 2014.

[4] T. Wang, Z. Liu, and Y. Chen, Load Balancing Task Scheduling Based on Genetic Algorithm in Cloud Computing, in *Proceedings of 12th International Conference on Dependable, Autonomic and Secure Computing*, 2014: 146-152.

[5] Z. Zhu and Z. Du, Improved GA-based task scheduling algorithm in cloud computing, *Computing Engineering and Applications*, 49(5):77-80, 2013.

[6] P. Babu and T. Amudha, A novel genetic algorithm for effective job scheduling in grid environment, in *Computational Intelligence, Cyber Security and Computational Models*, M. Senthilkumar, V. Ramasamy, S. Sheen, C. Veeramani, A. Bonato, and L. Batten, Eds. New Delhi: Springer India, 2014: 385-393.

[7] T. Pencheva and M. Angelova, Purposeful Model Parameters Genesis in Multi-population Genetic Algorithm. *Global J Technol Optim*, 5(1): 164, 2014.

[8] ST. Maguluri, R. Srikant, and L. Ying, Stochastic models of load balancing and scheduling in cloud computing clusters, in *2012 Proceedings IEEE INFOCOM*, 2012: 702-710.

[9] L. Huang, A Research on Task Scheduling Algorithm of Cloud Computing Based on Genetic Algorithm. Xiamen: *Xiamen University*, 2014.

[10] C. Huang, D. Hu, and X. Yu, Application of multiple-objective genetic algorithm for task scheduling in cloud environment. *Information Technology*, 38(5): 130-134, 2014.