A Hybrid genetic scheduling strategy

Benting Wan
software institute, Jiangxi University of finance and economics, Nanchang, Jiangxi,China
ren_btw@163.com

Abstract

A hybrid genetic scheduling strategy (H-GA) is described in this article, H-GA combines with grouping and load balancing strategy based on traditional genetic algorithm (GA). First, tasks are divided into several different subgroups by task granularity. Then, task subgroup which is selected by granularity from big to small is used to schedule by the genetic algorithm, and during scheduling, the load balancing strategy is used to adjust task distribution in the individual. Grouping can cut down the length of individual, which speeds up convergence of genetic algorithm. Load balancing strategy can make the individual better, which also speeds up convergence of genetic algorithm. The implementation shows that converging speed of H-GA is faster than GA, and result of H-GA is optimal than GA if the iteration times are equal.

1. Introduction

It is proved that task scheduling of Grid environment is a NP problem[1]. And there are no scheduling algorithm fitting to various environments. To some systems, if there are many big granularity tasks needed to be executed and the result of scheduling algorithm is not optimal[4][5], the completing time will be badly affected, such as Min-Min algorithm [4]. However, genetic algorithm is a good solution for it. Presently, there are different genetic algorithms used to assign tasks[6][7][8][9][10]. When genetic algorithm is compared with simulating anneal and ant colony algorithms etc, the result of genetic algorithm is better, but genetic algorithm also has many inherent defects: slow converging speed, premature convergence, lack of climbing ability and so on, and the defect of premature convergence is the most serious one [10]. In order to avoid these defects, people have proposed many advanced genetic algorithms, some of them improve some phases of GA, and the others of them are combined with other methods. To the first one, the common methods are used: encoding, individual and population initialing, fitness functions, selection, crossover and mutation methods are improved on[2][6][11][12][17]. To the second one, GA often combines with local searching, simulating anneal, tabu and some other algorithms[13][14][15][16].

When genetic algorithm is used to schedule tasks, the length of individual will be too long if there are too much tasks, and the solution space will be much bigger, therefore the converging speed will be much more slowly[10]. However, grouping strategy can reduce solution space to improve GA performance. It is obvious that good individual can accelerate the convergence of GA, the load balancing strategy is used to optimize individual; after mutation is transacted the load balancing strategy is used to adjust distribution of the tasks in the individual so as to keep the load much more balanced.

The H-GA meets the following conditions: tasks are independent, each computer performance is determined, communication conditions are invariable, and a large amount of the tasks need to be scheduled. The following section describes H-GA.

2. Hybrid genetic schedule strategy

2.1. Task grouping

If there are many tasks needing to be scheduled, the length of individual is too long, which makes GA converging speed very slow. In order to improve GA converging speed, the individual is divided into some subgroups. But if there are too many subgroups, the number of tasks are few to each subgroup, on the one hand, the solution is sub-optimal for each subgroup, which affects the performance of the whole solution; on the other hand, the converging time are seriously affected if there are too many subgroups because each subgroup undergoes each phase of GA. So appropriate grouping which can reduce GA executing time is very important to GA. Task grouping strategy is given below.

(1) All tasks are sorted in descent order by granularity.

(2) Then tasks are divided into some subgroups, the number of subgroups can be calculated by formula(1).

$$groups = tasks /(computers * avgtask) \qquad (1)$$

The *tasks* presents how much tasks need to be assigned, and the *computers* presents how much computers take part in computation, and the *avgtask* presents average number of tasks assigned to each computer. The tasks and computers are determined by heterogeneous environment, and the *avgtask* is given by users, generally, *avgtask* arranges from 8 to 512. If the *avgtask* is very big, the number of groups is very little. The number of subgroups is 4 about implement of this paper.

(3) Task subgroups are handed in GA from big to small by granularity that can make computer load much more balance.

2.2. Hybrid Genetic Algorithm

The processes of H-GA include five steps: encoding, individual and population initialing; fitness value of individual computing; selection, crossover, mutation; load balancing transaction and stopping condition. They are described following.

(1) Encoding, individual and population initialing

Each individual in the population represents a possible schedule. Fig.1 shows the encoding used. Each character is a mapping between a task and computer. Each character contains the unique identification number of a task, with 0 being used to delimit different computer queues, where P_i is computer i.

3 6	0	1 8 5	0	2 7 9	0	4

Fig. 1 Encoding of genetic algorithm

The number of individuals is half of computers, and tasks stochastically distributed among individual, and the number of tasks is also stochastically distributed in one computer.

(2) Fitness function

A fitness function attaches a value to each individual in the population, which indicates the goodness of the schedule. Here, relative load is used to generate fitness values. After tasks are mapped to computer *i*, the executing time of computer *i* can be calculated by formula(2).

$$S_i = L_i + \sum_{k_j=0}^{N_i} t_{ik_j} + \sum_{k_j=0}^{N_i} Com_{ik_j} \qquad (2)$$

While S_i denotes executing time of computer *i*, L_i denotes current load of computer *i*, and N_i denotes the number of tasks mapped to computer *i*, t_{ik_j} denotes task k_j computing time in the i^{th} computer. And Com_{ik_j} denotes communication time from computer k_j to *i*. If the executing time overlaps to communicating time, the Com_{ik_j} is zero, and N_i should satisfy formula(3).

$$N = \sum_{i=1}^{m} N_i \qquad (3)$$

Then absolute value of the subtraction of two computers load is presented as formula (4). The *m* denotes the number of computers.

$$P_i = \left| S_{(i+1)\%\, m} - S_i \right| \qquad (4)$$

Then fitness value of each individual can be calculated by formula(5).

$$E_i = \sum_{i=1}^{M} P_i \qquad (5)$$

If E_i is small, load balancing of computers is very well, so the individual is better.

(3) Selection, crossing, mutation

We choose to use the standard weighted roulette wheel method of selection which is widely used by previous researchers who have applied genetic algorithm to task scheduling [2]. In order to use roulette wheel method, the E_i need to be transformed according to formula(6).

$$F_i = \frac{1}{E_i} \qquad (6)$$

The F_i is bigger, the better the individual is. Each individual *i* is assigned a slot between 0 and 1. The value of slot *i* can be calculated by formula(7).

$$\delta_i = \frac{F_i}{\sum_{j=1}^{m} F_j} \qquad (7)$$

$$\sum_{i=1}^{P} \delta_i = 1 \qquad (8)$$

And *p* is the number of individuals in population. After selection process is complete, cycle crossover method is used to promote exploration[2][12]. The method of mutation to promote exploration of the search space is that we randomly swap elements of a randomly chosen individual in the population.

(4) Load balancing transaction

The main object of task scheduling in heterogeneous environment is to make load of each computer basically balanced, which can make tasks completed in least time. Therefore, whether or not the individual is chosen depends on the load balancing status of individual, if load keeps balanced to each computer, the individual is chosen. However, the goodness of individual is related to the converging speed of generic algorithm, in order to make individual evolve better, load balancing strategy is described by Fig.2.

Maxload presents Maximal load and *Maxcom* presents Maximal computer, *Minload* presents Minimal load and *Mincom* presents minimal computer. $Selecttask_{maxcom}$ presents task *Selecttask* executing time on *Maxcom*, and $Selecttask_{Mincom}$ presents task *Selecttask* executing time on *Mincom*.

```
For j=1 to m/2
Begin
For I=1 to k       // times of individual nee to be transacted.
   Begin           // adjust task distribution according load of gene
      Find out two computers in individual, one is Maxcom,
      Which load is Maxload, another is Mincom, which load is Minload.
      Randomly chooses a task in Maxcom, the task is present Selecttask.
      Maxload subtract Selecttask_maxcom executing time, Minload add
      Selecttask_mincom executing time
      // Compare load adjust task distribution
      If Minload still less than Maxload
         Delete the task from Maxcom and insert it into Mincom
         Update the load of Maxcom and Mixcom.
      Else
         Break     // if task distribution satisfy requirement,
   End    // one individual is transaction.
End
```

Fig. 2 Load balancing strategy

From fig.2, the load balancing strategy includes six step, they are described by the following.

(1) Find out two computers in an individual. One has the maximal load, and another has the minimal load.

(2) Randomly chooses a task in *Maxcom*.

(3) First, *Maxload* subtract $Selecttask_{maxcom}$ executing time, then *Minload* add $Selecttask_{mincom}$ executing time; if *Maxload* still less than *Minload*, then delete the task from *Maxcom* and insert it into *Mincom*.

(4) Update the load of *Maxcom* and *Mixcom*.

(5) Repeat k times (1) to (3) step.

(6) Apply the same load balancing transaction to other individuals until all individuals are assigned.

One aspect, load balancing transaction can make individual evolve better, and speeds up convergence of GA and reduce count of iteration. Another aspect, when transaction counts increase, the time of load balancing transaction will increase. *Sum* denotes time complexity of load balancing transaction, *Sum* is calculated by formula (9), k denotes the number of load balancing transaction times, *generation* denotes iterated times, *chrom* denotes number of individuals, m denotes number of computers.

$$Sum = O(m \times chrom \times k \times generation) \quad (9)$$

To H-GA, it needs to iterate *generation* times, and there are *chrom* individuals need to be transacted, and to each individual, k times need to be transacted, and m computers need to be compared in a individual. So the genetic algorithm complexity can be presented formula (9).

The implement reflects the affection of load balancing transaction count for converging time of algorithm. In this algorithm, the reasonable transaction counts are $k = m$.

(5) Stopping condition

Stopping conditions are that E_i less than m or the iteration times less than 1000, the m denotes the number of computers.

3. Implement

Implement environment includes: one computer runs scheduling strategy, eight computers execute tasks assigned. The task granularity is randomly produced, and maximal granularity needs 10 minutes to compute. Number of tasks arrange from 512 to 4096. In order to compare scheduling algorithm performance, there are several results are given.

First, transaction times of loading balance affect algorithm performance, when the transaction number equal the number of computers, the genetic scheduling algorithm is optimal (Fig.3). When the transaction times increase, the algorithm converging speed decreases.

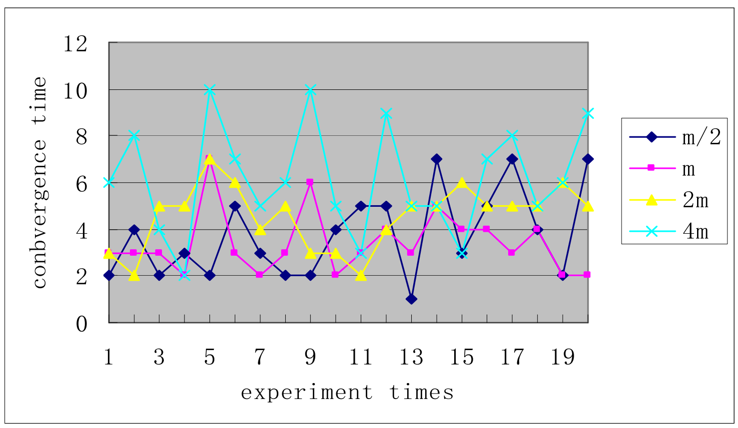

Fig. 3 Transaction times effects algorithm performance

Second, Grouping tasks also affect algorithm converging time. From the Fig.4, if the tasks do not use grouping strategy, the converging time fluctuates greatly. When the 4096 tasks are divided into 4 groups, the algorithm is optimal and scheduling strategy average executing time dose not exceeds 4 seconds (Fig.4).

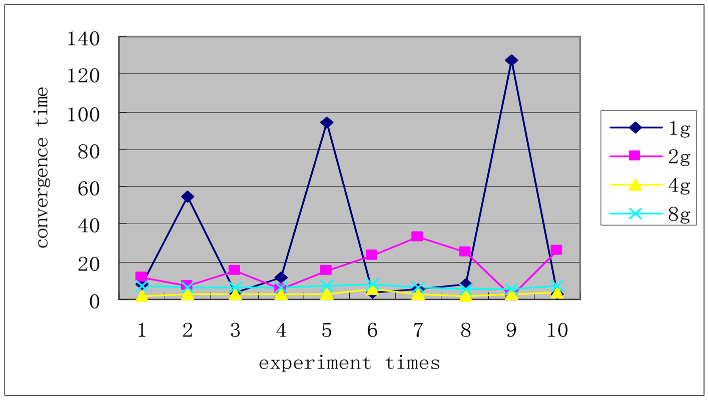

Fig. 4 Grouping affects algorithm converging time

Third, Grouping affects the result of algorithm. Task grouping affects the subtraction of maximal and minimal load of computers is given Fig.5. From the Fig.5, if algorithm does not use grouping strategy, it often occurs emanative. When the 4096 tasks are divided 4 groups, the H-GA is optimal and subtracting of maximal load and minimal load do not exceed 3 seconds.

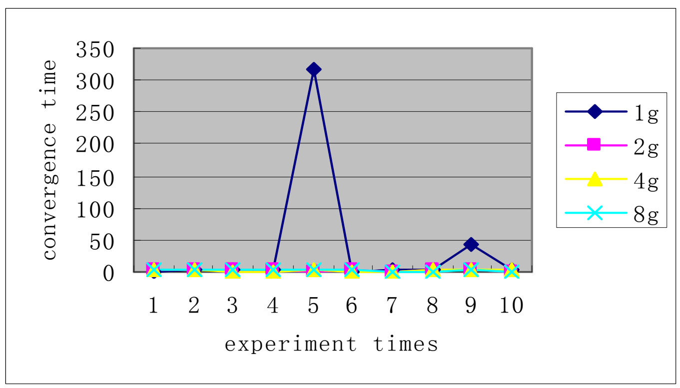

Fig. 5 Grouping affects the result of algorithm

Forth, loading balancing affects algorithm converging result given by the Fig.6. If the genetic algorithm does not use load balancing strategy, the algorithm is emanative, but load balancing strategy can make genetic algorithm convergent within 1000 times.

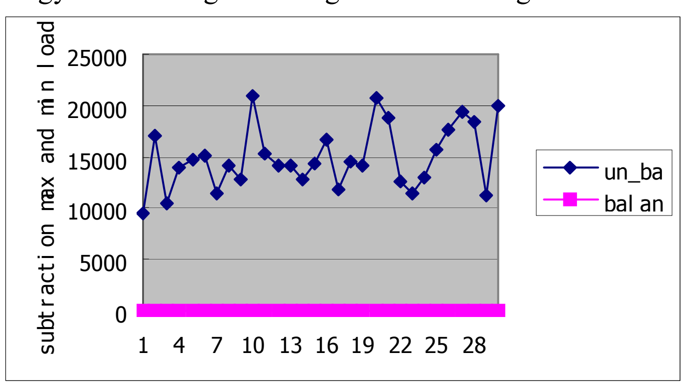

Fig. 6 Load balancing strategy effect.

According to test value that the scheduling strategy is repeated 250 times, when the 4096 tasks are divided into 4 subgroups and load balancing transaction times equals the number of computers, the scheduling strategy is convergent and the average iteration times of subgroups does not exceed 300.

4. Conclusion

The H-GA combines with grouping and load balancing strategy. Grouping strategy divides tasks into subgroups, it is trade-off, it does reduce the genetic algorithm computing time through cutting down individual length, but it increases genetic algorithm computing time and reduces genetic algorithm searching space because of many subgroups. The implement shows when 4096 tasks are divided into 4 groups, H-GA average converging time is the shortest one and the result of genetic algorithm is sub-optimal. The implement also shows that load balancing transaction greatly speeds up genetic algorithm convergence, if the genetic algorithm does not use load balancing strategy, the genetic algorithm is basically emanative after it iterates about 1000 times. So H-GA has better performance than traditional genetic algorithm.

Reference

[1] R.L. Graham, L.E. Lenstra, J.K. Lenstra, and A.H.Kan. Optimization and Approximation in Deterministic Sequencing and Scheduling. A survey, in Annual Discrete Mathematics, 1979 :287-326.

[2] I. M. Oliver, D. J. Smith, and J. Holland. A study of permutation crossover operators on the traveling salesman problem. In Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application, Lawrence Erlbaum Associates, Inc. 1987 :224–230.

[3] Andrew J. Page and Thomas J. Naughton. Dynamic task scheduling using genetic algorithms for heterogeneous distributed computing. 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 6, 2005: 189a.

[4] DING Qing, CHEN Guo-Liang, GU Jun. A Unified Resource Mapping Strategy in Computational Grid Environments. Journal of Software. 2001; 13(7):1303-1308.

[5] HE Xiaoshan, Xian-He Sun, Gregor von Laszewski. QoS Guided Min-Min Heuristic for Grid Task Scheduling. Journal of Computer Science &technology. 2003; 18(4):442-451.

[6] A. Y. Zomaya, C. Ward, and B. Macey. Genetic scheduling for parallel processor systems: comparative studies and performance issues. IEEE Transactions on Parallel and Distributed Systems, August 1999;10(8):795–812.

[7] Easton, F, Mansour, N, A Distributed Genetic Algorithm For Deterministic And Stochastic Labor Scheduling Problems, European Journal of Operational Research, 1999:505-523.

[8] Ricardo C. Correa, Afonso Ferreira, and Pascal Rebreyend. Scheduling Multiprocessor Tasks with Genetic Algorithms. IEEE Transactions on Parallel and Distributed Systems, Aug. 1999 ;10(8): 825-837.

[9] Albert Y. Zomaya, Chris Ward, and Ben Macey. Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues. IEEE Transactions on Parallel and Distributed Systems. 1999;10(8):795-812.

[10] Xing WenXun, Xie JinXing. Modern optimal Computing Methods. TsingHua Publisher. 2003 :140-188.

[11] Limin Han, Graham Kendall. Guided Operators for a Hyper-Heuristic Genetic Algorithm. Advances in Artificial Intelligence: 16th Australian Conference on AI, Perth, Australia, December 3-5, 2003. Proceedings. Springer Berlin / Heidelberg. 2003 :807 – 820.

[12] Cowling, P.I., Kendall, G., Han, L., An Investigation of a Hyperheuristic Genetic Algorithm Applied to a Trainer Scheduling Problem. Accepted for 2002 Congress on Evolutionary Computation (CEC2002), Hilton Hawaiian Village Hotel, Honolulu, Hawaii, May 2002 :12-17.

[12] Cowling, P.I., Kendall, G., Han, L., An Investigation of a Hyperheuristic Genetic Algorithm Applied to a Trainer Scheduling Problem. Accepted for 2002 Congress on Evolutionary Computation (CEC2002), Hilton Hawaiian Village Hotel, Honolulu, Hawaii, May 2002 :12-17.

[13] Limin Han, Graham Kendall, Peter Cowling. An Adaptive Length Chromosome Hyperheuristic Genetic Algorithm for a Trainer Scheduling Problem. SEAL2002: 267-271.

[14] Limin Han, Graham Kendall. An Investigation of a Tabu Assisted Hyper-Heuristic Genetic Algorithm. Evolutionary Computation, CEC '03. 2003: l.3:2230- 2237.

[15] John Conner, Yuan Xie, Mahmut Kandemir, Robert Dick, and Greg Link. FD-HGAC: A Hybrid Heuristic Genetic Algorithm Hardware/Software Co-synthesis Framework with Fault Detection. ASPDAC 2005 Digital Object Identifier. 2005;2: 709- 712.
[16] Sankaran Prashanth, Daniel Andresen. Using implicit fitness functions for genetic algorithm-based agent scheduling. Proceedings of the 2001 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2001), Las Vegas, June 26-29, 2001: 445--450.
[17] A. Y. Zomaya and Y.-H. Teh. Observations on using genetic algorithms for dynamic load-balancing. IEEE Transactions on Parallel and Distributed Systems, September 2001; 12(9):899–911.

Authors

Benting wan is an instructor in the software institute, Jiangxi University of finance and economics. He received his Ph.D. degree in computer application technology at the China University of Petroleum(beijing) in 2006. His current research interests include distributed &parallel computing and mobile computing.