PGGA: A Predictable and Grouped Genetic Algorithm for Job Scheduling

Maozhen Li and Bin Yu
School of Engineering and Design, Brunel University,
Uxbridge, UB8 3PH, UK
{Maozhen.Li, Bin.Yu}@brunel.ac.uk

Man Qi
Dept of Computing, Canterbury Christ Church University College,
Canterbury, Kent, CT1 1QU, UK
mq4@canterbury.ac.uk

Abstract

This paper presents a predictable and grouped genetic algorithm (PGGA) for job scheduling. The novelty of the PGGA is twofold: (1) a job workload estimation algorithm is designed to estimate a job workload based on its historical execution records, (2) the divisible load theory (DLT) is applied to predict an optimal solution in searching a large scheduling space so that the convergence process can be speeded up. Comparison with traditional scheduling methods such as first-come-first-serve (FCFS), random scheduling and a typical genetic algorithm (TGA) indicates that the PGGA is more effective and efficient in finding optimal scheduling solutions.

Keywords: Job Scheduling, Job Workload Estimation, Divisible Load Theory, Predictable Genetic Algorithm, Load Balancing.

1. Introduction

The Grid [1] is evolving as a promising computing platform for engineers and scientists to solve data and computation very intensive problems. Job scheduling is an indispensable part of the Grid. One major role of a job scheduling system is to minimum the total execution time (makespan) of the jobs submitted to a Grid environment. Well-known workload management systems such as Condor [2] and Sun Grid Engine [3] have built-in simple job scheduling mechanisms such as first-come-first-server (FCFS), random scheduling, priority-based scheduling, but they do not have comprehensive heuristics methods for job scheduling. In a Grid environment, many users may concurrently submit jobs for execution. The scheduling of m jobs to n resources, which is named as an m/n type scheduling, becomes a NP-hard problem [4]. Genetic algorithms (GAs) have been widely used in this area trying to find optimal or near optimal scheduling solutions.

When perform m/n type scheduling, traditional GAs generate an optimal or a near optimal scheduling solution by ranking the fitness value of each chromosome in an evolution process. One major problem with these GAs is that they cannot dynamically predict an optimal fitness value based on historical execution records of the jobs to be scheduled. Therefore, the fitness of a scheduling solution produced by a GA is largely dependent on the number of generations (iterations) used in the evolution process. While a large number of iterations could have a higher probability in producing a better scheduling solution than a small number of iterations, the cost is that the former method usually takes longer to complete the evolution process than the latter method. A good job scheduling approach should be quick enough to produce an optimal or a near optimal scheduling solution.

In this paper, we present a predictable and grouped genetic algorithm (PGGA) for m/n type scheduling. The PGGA is novel in two aspects. On one hand, it uses a job workload estimation algorithm to estimate a job workload based on its historical execution records. On the other hand, the PGGA uses the divisible load theory (DLT) [5] to dynamically predict an optimal fitness value during its runtime which can speed up the convergence process in finding an optimal scheduling solution.

The remainder of this paper is organized as follows. Section 2 presents a job workload estimation algorithm. Section 3 introduces the PGGA, which is extended from an implementation of a typical genetic algorithm (TGA). Section 4 evaluates the performance of the PGGA in scheduling a large number of jobs. Both theoretical and practical tests have been performed. Section 5 gives related work on job scheduling with GAs, and Section 6 concludes this paper.

2. A Job Workload Estimation Algorithm

Jobs considered in the algorithm are independent and indivisible. Computing nodes used in a cluster environment are dedicated and have the same architecture but with different computing capabilities. Each job has an input with a certain data size and a unique job name. We only consider the computing capability as the performance metric of a computing node in the algorithm. Furthermore, the communication cost to send a job from one computing node to another is not considered in the algorithm as the network bandwidth in a cluster environment is normally fixed.

Let

- J_k be the name of job k, k∈{1, 2, . . ., m}.
- $J_k(d)$ be a job J_k with an input data size of d.
- $T(J_k(d))$ be the number of normalized time units needed to complete the execution of $J_k(d)$.
- P_i be a computing node, i∈{1, 2, . . . , n}.
- $F(P_i)$ be the computing capability of node P_i .
- $U(P_i)$ be the current utilization rate of node P_i.
- $W(J_k(d))$ be the estimated workload of $J_k(d)$.

For a job $J_k(d)$ to run on a node P_i, we have

$$T(\mathrm{J}_k(d)) = \frac{W(J_k(d))}{U(P_i)\times F(P_i)} \Rightarrow W(J_k(d)) = T(J_k(d))\times U(P_i)\times F(P_i) \qquad (1)$$

Let

- $LS_k(W(J_k(d_a)), W(J_k(d_b)))$ represent the slope of the line $((W(J_k(d_a)),d_a), (W(J_k(d_b)),d_b))$.
- $R=\{(LS_1)^*, (LS_2)^*, . . ., (LS_k)^*, . . ., (LS_m)^*\}$.
- $X=|d_a - d_b|$, $Y=| W(J_k(d_a)) - W(J_k(d_b))|$.
- H be a historical job execution record set.
- Q_1 be a queue for jobs without estimated workload values.
- Q_2 be a queue for jobs with estimated workload values.

Then we have

$$LS_k(W(J_k(d_a)), W(J_k(d_b))) = \frac{Y}{X} \qquad (2)$$

The job workload estimation algorithm is given in Figure 1. A specific job with more execution records in the list R can have a more accurate estimated workload using the algorithm.

```
1:  Receive a new job $J_k$ with an input data size of $d_c$
2:    if the job $J_k$ has execution records in the list R, then
3:      if $d_c$ is in the range of any $(d_a, d_b)$ in R, then
4:        Calculate an estimated workload $W_e(J_k(d_c))$ of the job $J_k(d_c)$ based on formula
              $W_e(J_k(d_c)) = LS_k(W(J_k(d_a)),W(J_k(d_b)))\times d_c$
5:        Put the job $J_k(d_c)$ with the estimated workload in queue $Q_2$
6:        Update R
7:          Replace $LS_k(W(J_k(d_a)),W(J_k(d_b)))$ with $LS_k(W(J_k(d_a)),W(J_k(d_c)))$ and
              $LS_k(W(J_k(d_c)), W(J_k(d_b)))$ once the job $J_k(d_c)$ finishes its execution, where $W(J_k(d_c))$
              is the real workload of job $J_k(d_c)$
8:        Go to step 1
9:      endif
10:     else if $d_c$ is not in the range of any $(d_a, d_b)$ in R, then
11:       Find the closest $d_a$ to $d_c$, and then find the $d_b$ from the record $LS_k(W(J_k(d_a)),W(J_k(d_b)))$ in R
12:       Calculate an estimated workload $W_e(J_k(d_c))$ of the job $J_k(d_c)$ based on formula
              $W_e(J_k(d_c)) = LS_k(W(J_k(d_a)),W(J_k(d_b)))\times d_c$
13:       Put the job $J_k(d_c)$ with the estimated workload in queue $Q_2$
14:         Update R
15:           Add $LS_k(W(J_k(d_a)),W(J_k(d_c)))$ in R once the job $J_k(d_c)$ finishes its execution, where
              $W(J_k(d_c))$ is the real workload of job $J_k(d_c)$
16:       Go to step 1
17:     endif
18:   endif
19:   else if there are two execution records of job $J_k$ in H , e.g., $W(J_k(d_a))$ and $W(J_k(d_b))$, then
20:     Calculate $LS_k(W(J_k(d_a)),W(J_k(d_b)))$ based on formula (2)
21:     Add $LS_k(W(J_k(d_a)),W(J_k(d_b)))$ in R
22:     Delete the two records from H
23:     Go to step 2
24:   endif
25:   else if there is no or just one execution record of job $J_k$ in H, then
26:     Put the job $J_k(d_c)$ in queue $Q_1$
27:     Add a new execution record with the real workload $W(J_k(d_c))$ of job $J_k(d_c)$ in H once the job
              $J_k(d_c)$ finishes its execution
28:     Go to step 1
29:   endif
```

Figure 1. A job workload estimation algorithm.

3. Scheduling Algorithm Design

3.1 A Typical Generic Algorithm for Job Scheduling

To solve an optimisation problem, a GA solution needs to be represented as a chromosome encoded as a set of strings, which are normally binary strings. However, for the problem of job scheduling, a binary representation is not feasible because the number of jobs for scheduling can increase dramatically which results in a long binary string. In the TGA, we employ decimal string to represent a chromosome. Computing nodes and jobs are uniquely numbered. As shown in Figure 2, a computer node is represented as a gene in a chromosome. The position of a gene represents the sequence number of a job. The sequence of jobs is organized in an ascending order from the left side to the right side corresponding to a chromosome.

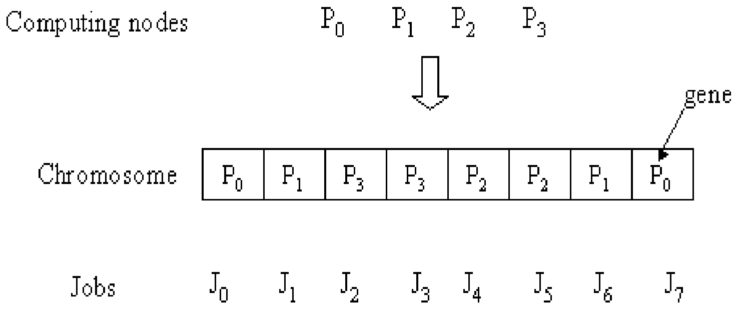


Figure 2. Problem representation in the TGA.

Each chromosome is associated with a fitness value which will be calculated by the following fitness function.

Let

- n be the number of computing nodes.
- T_i be the number of normalized time units needed to finish all the jobs on a computing node P_i, where $i \in \{1, 2, . . ., n\}$.
- $m(P_i)$ be the number of jobs allocated to the node P_i.
- C_K be the Kth chromosome in one chromosome population.

Then, the fitness function can be defined as follows:

$$Fitness(C_K) = \frac{1}{Max(T_1, T_2, \ldots, T_n)} \tag{3}$$

According to formula (1), $T_i = \sum_{k=1}^{m(Pi)} \frac{W(J_k)}{U(P_i) \times F(P_i)}$

Each chromosome in one population is ranked using its associated fitness values calculated by formula (3). From the testing results to be presented in Section 4, we will see that other scheduling methods such as FCFS may produce a better solution than the TGA. This is because the TGA may produce a reasonable good scheduling solution with a specified number of iterations rather than an optimal solution. Moreover, as a typical implementation of a GA, the TGA also has a slow convergence process.

3.2 A Predictable and Grouped GA for Job Scheduling

Two improvements have been made in the PGGA. On one hand, computing nodes are classified into groups based on their computing capabilities. All nodes in the same group have the same computing capability. On the other hand, the DLT is applied in the PGGA to predict an optimal fitness value. According to the DLT, taking only the computation into account, an optimal solution to schedule m jobs to multiple groups of computing nodes with an objective to reach a minimum makespan is that each group should finish their jobs at the same time. Moreover, each computing node in the same group should finish their jobs at the same time too.

Let

- N be the number of groups classified.
- G_i be the ith group.

- $N(G_i)$ be the number of computing nodes in group G_i, $i \in \{1, 2, \ldots, N\}$.
- $W(G_i)$ be the total estimated workloads of jobs in group G_i.
- W be the total estimated workloads of jobs in the N groups.
- $T(G_i)$ be a predicted number of normalized time units needed to finish all the jobs in group G_i.
- $F(G_i)$ be the utilizable computing capability of each node in group G_i, $F(G_i) = U(P_i) \times F(P_i)$ where P_i is a computing node in the ith group.

Then we have

$$W = \sum_{i=1}^{N} W(G_i) \tag{4}$$

According to formula (1) and the DLT, we have

$$T(G_i) = \frac{W(G_i)}{N(G_i) \times F(G_i)} \tag{5}$$

where $T(G_1) = T(G_2) = \ldots = T(G_n)$ (6)

Using formula (4), (5) and (6), we have

$$T(G_i) = \frac{W}{\sum_{i=1}^{N} (F(G_i) \times N(G_i))} \tag{7}$$

Here the $T(G_i)$ represents the minimum time units needed to schedule m jobs to n computing nodes. It is a predicted optimal fitness value for selecting chromosomes from one generation to the next generation. As parameters used in formula (7) such as $F(G_i)$, $N(G_i)$, N are known and W can be estimated, the value of the $T(G_i)$ can be calculated dynamically during the runtime of PGGA.

We reuse the fitness function (3) in the PGGA to select an optimal chromosome whose fitness value should be close enough to the reverse value of the predicted $T(G_i)$. The chromosomes whose fitness values are far away from the reverse value of the T(Gi) will be thrown away. From the testing results to be given in Section 4, we will see that PGGA speeds up the convergence process and can always produce a better scheduling solution compared with other scheduling methods such as FCFS, random and the TGA.

4. Performance Evaluation

We have done comprehensive tests to evaluate the performance of the PGGA for m/n type scheduling. Tests are classified into two classes - theoretical tests and practical tests. Theoretical tests are performed without using the job workload estimation algorithm as described in Section 2. Practical tests are performed in a Linux cluster environment using the job workload estimation algorithm.

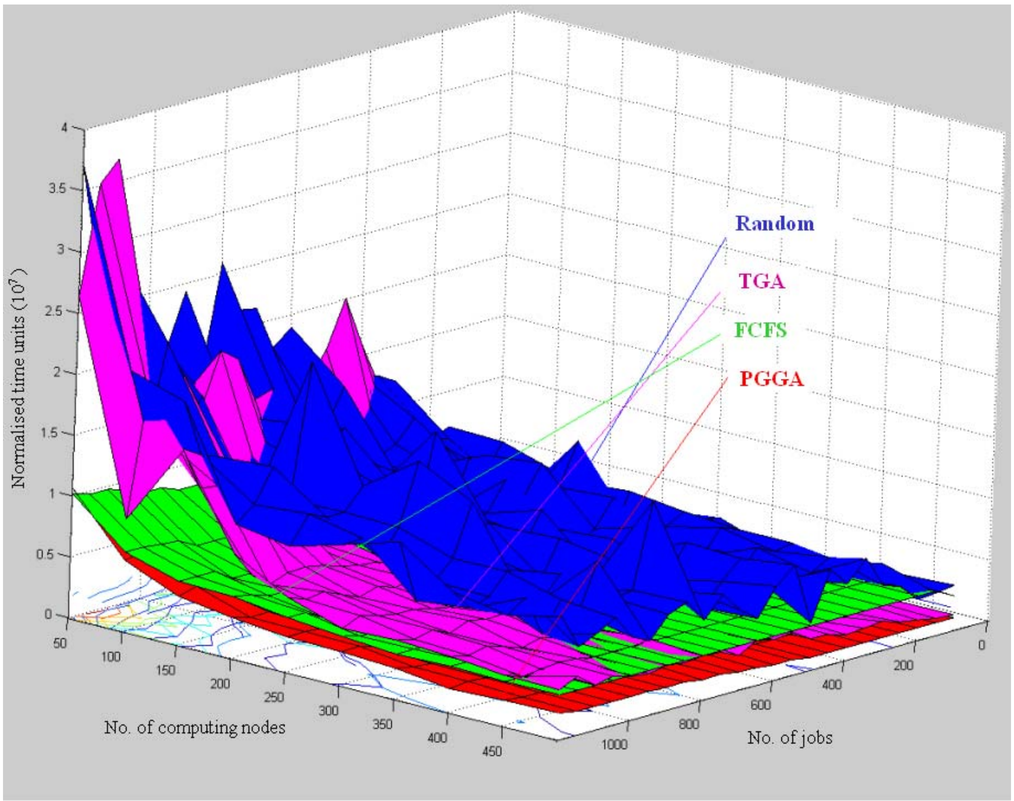


Figure 3. A performance evaluation of the PGGA.

4.1 Theoretical Tests

A job workload was randomly generated from 1-3,000,000 with a rate of 70% for small workload values and 30% for large workload values. We assumed five types of computing nodes – 2.8GHz, 2.6GHz, 2.0GHz, 1.8GHz, and 1.6GHz. A computing node of each type has an equal probability to be randomly chosen from the five types of computing nodes. We assumed the utilization of each computing node chosen is 100%. We define the time needed to finish one workload unit of a job on a computing node with a speed of 1GHz as one normalized time unit. The tests were performed in the following aspects.

4.1.1 Performance Evaluation of PGGA in Job Scheduling

We have done 240 tests to evaluate the performance of the PGGA for m/n type scheduling. The number of jobs started from 50 to 1200 with a step of 50. The number of computing nodes started from 50 to 500 with a step of 50. The maximum number of iterations used by both the TGA and the PGGA was 200.

As shown in Fig.3, the four curved surfaces represent the normalized time units needed to finish m jobs on n computing nodes using the four scheduling methods respectively. Among the four scheduling methods, the random approach produces the worst scheduling solutions and the PGGA is the best one. The performance of the PGGA is always better than that of FCFS and the TGA, as shown in Figure 4 and Figure 5 respectively.

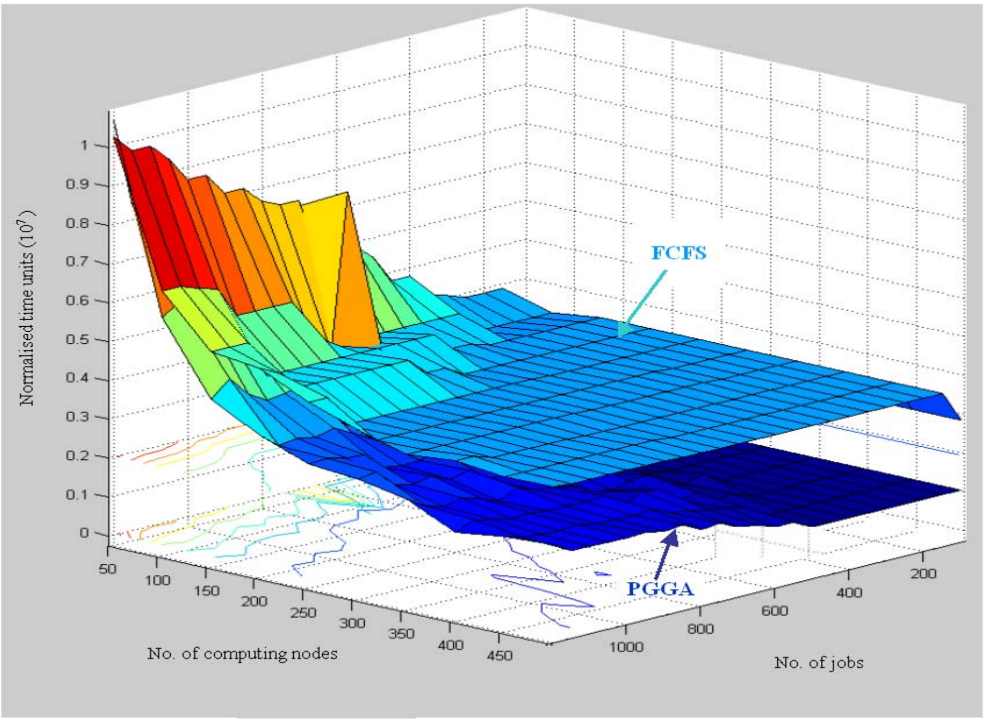


Figure 4. A performance comparison of FCFS and the PGGA.

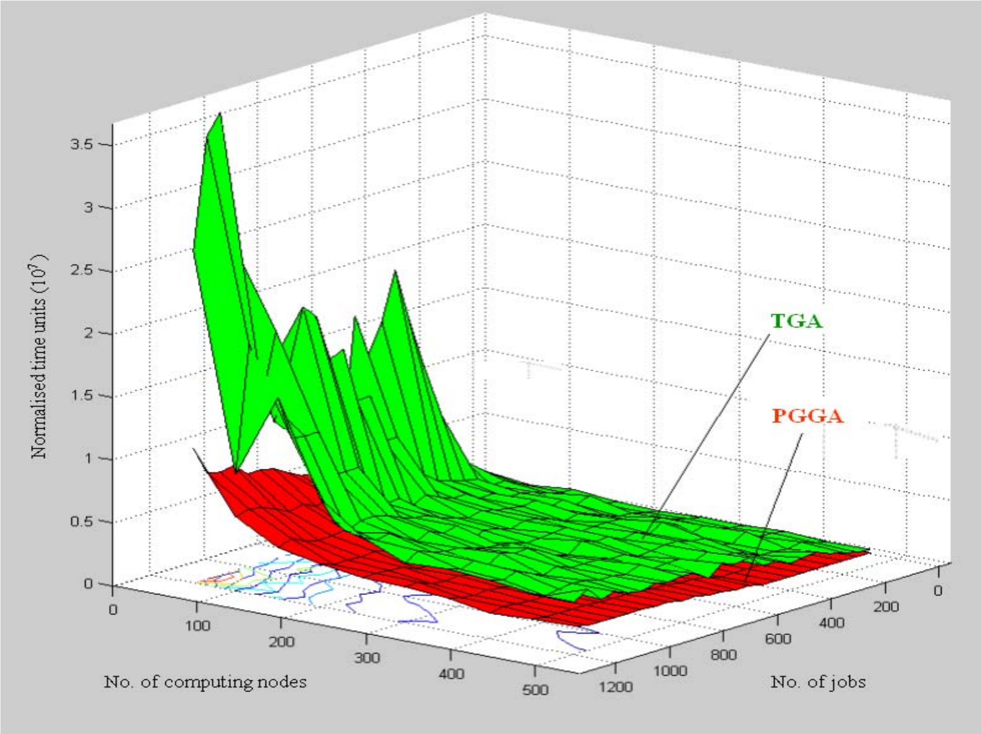


Figure 5. A performance comparison of the TGA and the PGGA.

However, the performance of the TGA is not always better than that of FCFS as shown in Figure 6. The reason is that the number of iterations used by the TGA was

200, which is not large enough to search all the scheduling spaces to find an optimal or near optimal solution. The larger the number of iterations is used in the TGA, the better a solution can be produced at the cost of more time units to be consumed. Figure 6 also emphasizes that one major problem with classical genetic algorithms in job scheduling is that they cannot guarantee an optimal solution using a reasonable number of iterations in the evolution process.

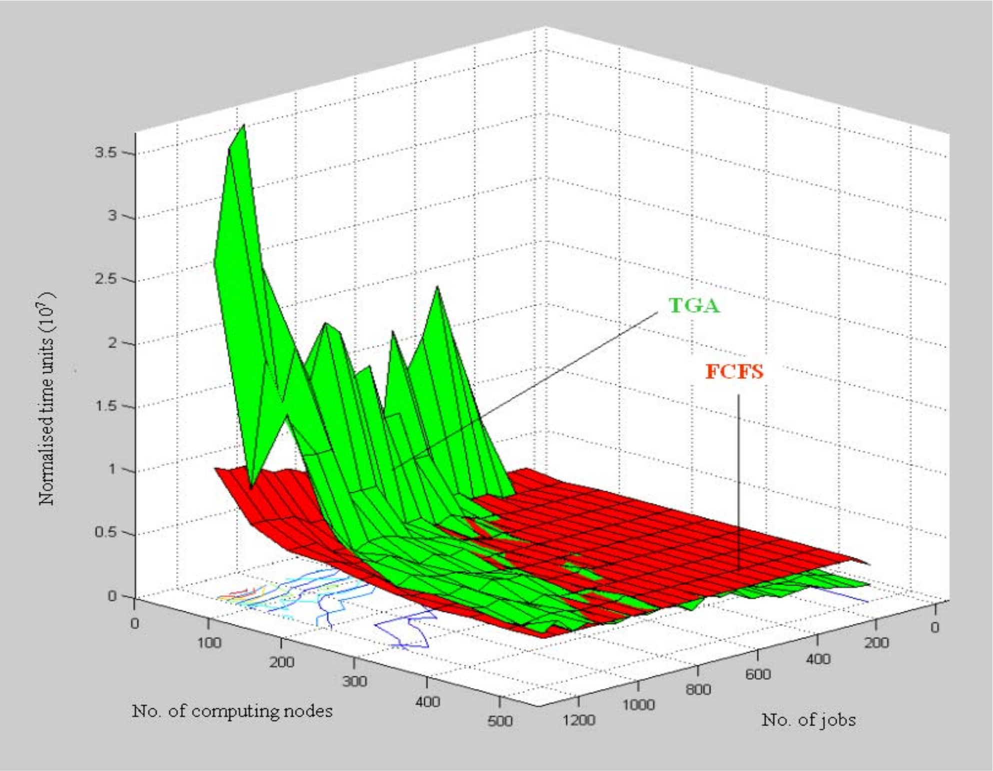


Figure 6. A performance comparison of FCFS and the TGA.

4.1.2 Convergence Evaluation of the TGA and the PGGA

From the tests described in Section 4.1.1 we know that the performance of PGGA is always better than that of the TGA in generating scheduling solutions. In this section, we compare the two algorithms in terms of convergence.

We randomly generated 1500 jobs with 10 different workload units, and used 500 computing nodes. The TGA and the PGGA were used to find solutions for scheduling the 1500 jobs to the 500 computing nodes. To test the convergence of the two algorithms, 10 tests were performed with the number of iterations for both the TGA and PGGA starting from 50 to 500. For each test, a scheduling solution was generated which was used to calculate the normalized time units for scheduling the 1500 jobs to the 500 computing nodes.

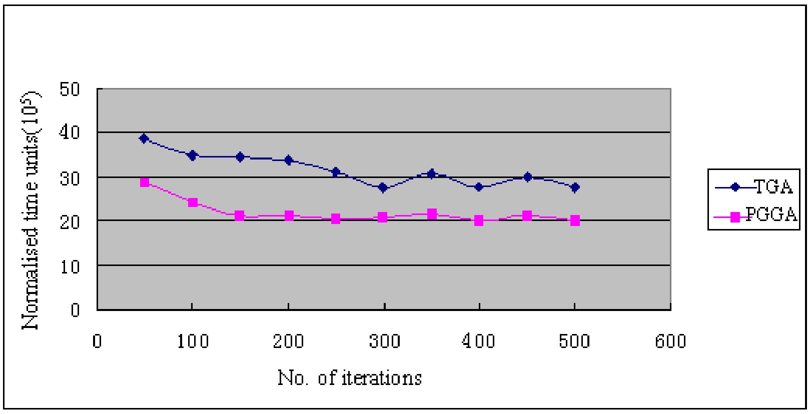
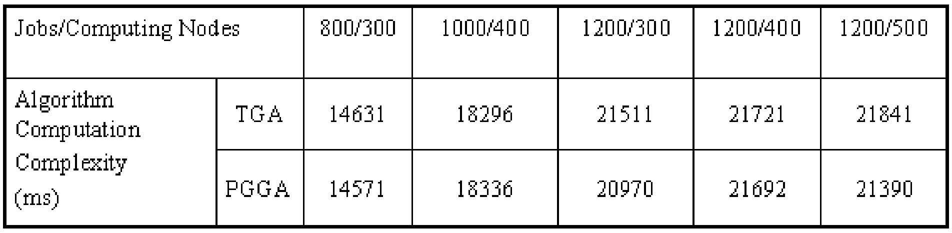


Figure 10. A convergence comparison of the TGA and the PGGA.

As shown in Figure 10, the PGGA has a faster convergence than the TGA, e.g., the PGGA only needs 150 iterations to converge, but the TGA needs at least 300 iterations to converge. Apart from that, the PGGA can also generate better scheduling solutions than the TGA.

4.1.3 Algorithm Complexity Evaluation of the TGA and the PGGA

When we performed the 240 tests as described in section 4.1.1, we also measured the computation complexity of the two genetic algorithms. The two algorithms were tested on an Intel Pentium IIII computing node with a 2.0GHz CPU and 256MB RAM running Windows XP Professional. Part of the results is given in Table 1, showing that the PGGA is faster than the TGA when the number of jobs and computing nodes gets large. The reason is that the PGGA dynamically predicts an optimal fitness value to select the chromosomes and thus needs fewer iterations than the TGA to produce a new generation of chromosomes.

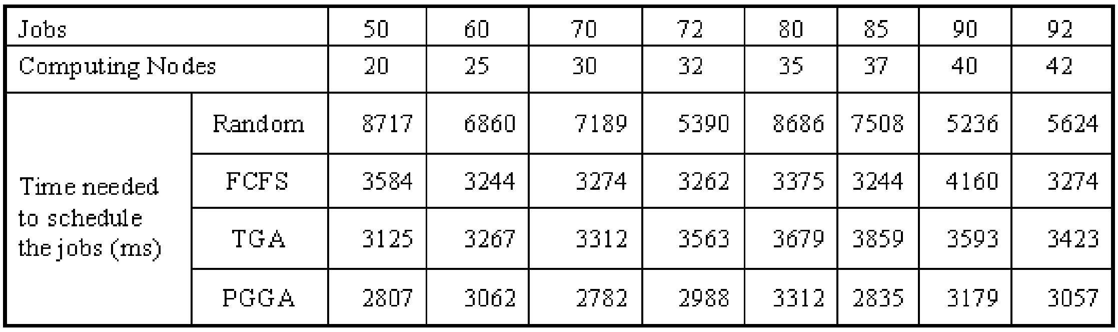


Jobs/Computing Nodes		800/300	1000/400	1200/300	1200/400	1200/500
Algorithm Computation Complexity (ms)	TGA	14631	18296	21511	21721	21841
	PGGA	14571	18336	20970	21692	21390

Table 1. The computation complexity of the TGA and PGGA.

4.2 Practical Tests

In this section, we present our testing results using the four methods for scheduling jobs in a Linux cluster environment, which had a maximum of 42 computing nodes with three types of computing speeds – Intel Pentium IIII 2.6GHz, Pentium IIII 2.0GHz and Pentium III 1.6GHz. Whereas the numbers of jobs and computing nodes were kept unchanged in one test, they were changed for different tests to represent a dynamic computing environment. The network bandwidth of the cluster is 100Mbps. Redhat Linux 7.3 and 9.0 were used in the cluster. The job workload estimation algorithm as described in Section 2 was used to estimate job workloads in the tests. The testing job was a loop to calculate two float values. An input data value (size) was used to specify the number the loops. The workload value of a job tested was in the range of 1-5. The time needed to finish a job with a workload value of 1 on a 2.6GHz computing node with a 100% utilization rate was 250ms. We performed 8 tests in the cluster environment to schedule jobs using the four methods. Table 2 shows the testing results for scheduling jobs in the Linux cluster environment taking only the computing capability of each node into account. Compared with FCFS, random and the TGA, the PGGA has again shown its superb performance in scheduling a large number of jobs in the cluster environment.



Jobs		50	60	70	72	80	85	90	92
Computing Nodes		20	25	30	32	35	37	40	42
Time needed to schedule the jobs (ms)	Random	8717	6860	7189	5390	8686	7508	5236	5624
	FCFS	3584	3244	3274	3262	3375	3244	4160	3274
	TGA	3125	3267	3312	3563	3679	3859	3593	3423
	PGGA	2807	3062	2782	2988	3312	2835	3179	3057

Table 2. Four methods for scheduling jobs in a Linux cluster environment.

5. Related Work

GAs have been widely used for static job scheduling [6,7,8], requiring the state of all jobs must be known a priori. This limits these schedulers to specific problems and systems. Dynamic GA schedulers [9,10,11] produce schedules at runtime which eliminates the necessity to know the properties of the jobs to be scheduled in advance, allowing for variable system and job properties to be considered. Our PGGA is a dynamic scheduler as it uses a job workload estimation algorithm to dynamically estimate a job workload by which to produce a predicted optimal fitness value for job scheduling.

Theoretically, a GA can find an optimal or a near optimal scheduling solution. However, it is not always the case when a GA is practically employed. The testing results presented in Section 4 have shown that a classical GA may produce a worse solution than a traditional FCFS scheduling method because cannot dynamically predict an optimal fitness value in the evolution process. The larger the number of iterations is used, the longer the time will be taken. A practical GA in job scheduling should have a fast convergence. One major research issue in applying GAs for job scheduling is to speed up the convergence process. Parallel GAs [12,13,14] have been proposed to solve this problem. However, these algorithms mainly distribute the computation complexity of a GA to multiple processors to speed up the convergence process instead of reducing the inherent complexity of a GA. Our PGGA makes use of the DLT to dynamically predict an optimal fitness value so that the computation complexity of a classical GA can be reduced in nature and the convergence process can be speeded up. Moreover, the PGGA can always produce an optimal or near optimal scheduling solution.

6. Conclusions

In this paper, we have presented PGGA, a predictable and grouped genetic algorithm for job scheduling in a cluster environment. The PGGA is novel in two aspects. On one hand, it uses a job workload estimation algorithm to dynamically estimate a job workload based on its historical execution records of the job. On the other hand, it makes use of the DLT to predict an optimal fitness value to speed up the convergence process. Comparison with traditional scheduling methods such as first-come-first-serve (FCFS), random scheduling and a typical genetic algorithm has indicated that the PGGA is more effective and efficient in finding optimal or near optimal scheduling solutions in terms of a large number of jobs and computing nodes.

Currently, jobs used by the PGGA are independent. However, the PGGA can be extended with a slight modification for scheduling dependent jobs, which can be grouped as one job as long as job dependencies are specified in a job submission. Future work will be focused on the extension of the PGGA to support dependent jobs. In addition, communication cost in job scheduling should also be taken into account in the extension of the PGGA to suit a Grid environment.

References

1. I. Foster and C. Kesselman, The Grid, blueprint for a new computing Infrastructure, (Mogan Kaufmann Publishers Inc., San Francisco, USA, 1998).
2. Condor, http://www.cs.wisc.edu/condor/
3. Sun Grid Engine, http://gridengine.sunsource.net/
4. M.R. Garey and D.S. Johnson, Computers and intractability, a guide to the theory of NP-completeness, (W.H. Freeman and Company, New York, 1979).
5. T.G. Robertazzi, Processor equivalence for daisy chain load sharing processors, IEEE Trans. Aerospace and Electronic Systems, 29 (4) (1993) 1216-1221.
6. I. Ahmad, Y.-K. Kwok, I. Ahmad, and M. Dhodhi, Scheduling parallel programs using genetic algorithms, in A. Y. Zomaya, F. Ercal, and S. Olariu eds, Solutions to parallel and distributed computing problems (John Wiley 2001) 231–254.
7. R. Correa, A. Ferreira, and P. Rebreyend, Scheduling multiprocessor tasks with genetic algorithms, IEEE Transactions on Parallel and Distributed Systems, 10(8) (1999) 825–837.
8. M. Grajcar, Genetic list scheduling algorithm for scheduling and allocation on a loosely coupled heterogeneous multiprocessor system, in Proc. of the 36th ACM/IEEE conference on Design automation (ACM Press, 1999) 280–285.
9. W. A. Greene, Dynamic load-balancing via a genetic algorithm, in Proc. Of the 13th IEEE International Conference on Tools with Artificial Intelligence (CS Press, 2001), 121–129.
10. M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, Dynamic mapping of a class of independent tasks onto heterogeneous computing systems, Journal of Parallel and Distributed Computing, 59(2) (1999) 107–131.
11. A. Y. Zomaya and Y.-H. The, Observations on using genetic algorithms for dynamic load-balancing, IEEE Transactions on Parallel and Distributed Systems, 12(9) (2001) 899–911.
12. A. D. Bethke, Comparison of genetic algorithms and gradient-based optimizers on parallel processors: efficiency of use of processing capacity, Tech. Rep. No. 197, University of Michigan, Logic of Computers Group, Ann Arbor, MI, 1976.

13. R. Hauser and R. Manner, Implementation of standard genetic algorithm on MIMD machines, in Y. Davidor, H. P. Schwefel and R. Manner, eds., Parallel problem solving from nature, PPSN III, (Springer-Verlag, Berlin, 1994) 504–513.
14. C.C. Pettey and M.R.Leuze, A theoretical investigation of a parallel genetic algorithm, in Proc. of the 3rd International Conference on Genetic Algorithms (Morgan Kaufmann , San Mateo, CA , 1989) 398–405.

Dr. Maozhen Li is a Lecturer in Distributed Systems at Brunel University, UK. He received the PhD from Institute of Software, Chinese Academy of Sciences in 1997. Dr. Li has published 30 papers in international journals and conferences. In 2004, Dr. Li co-authored a textbook on Grid Computing called The Grid: Core Technologies to be published by Wiley Publisher. Dr. Li's research interests are in the areas of Semantic Grid, Grid Workflow Management, Grid Portals, Resource Management and Job Scheduling, Grid Services Publication and Discovery.

Mr. Bin Yu is currently a PhD student in School of Engineering and Design at Brunel University, UK. His research interests are in the areas of Grid Services Wrapping, Publication, Discovery, and Resource Management and Job Scheduling.

Dr. Man Qi is a Lecturer in Dept of Computing, Canterbury Christ Church University College, UK. Dr. Qi was a Research Fellow in Dept. of Computer Science, University of Bath from Jan. 2001 to Oct. 2003. Her research interests are in the areas of Computer Graphics, Computer Animation, Multimedia and Grid Computing Applications.