An Improved Genetic Algorithm with Limited Iteration for Grid Scheduling

Hao Yin[1, 2], Huilin Wu[2], Jiliu Zhou[1, 2]
[1]*School of Computer Science, Sichuan University, No.24 South Section 1, Yihuan Road, Chengdu, 610065, China*
[2]*School of Electronic Information, Sichuan University, No.24 South Section 1, Yihuan Road, Chengdu, 610065, China*
yinhaoscu@hotmail.com, whl0912@hotmail.com, zhoujl@scu.edu.cn

Abstract

In Grid environment the numbers of resources and tasks to be scheduled are usually variable. This kind of characteristics of grid makes the scheduling approach a complex optimization problem. Genetic algorithm (GA) has been widely used to solve these difficult NP-complete problems. However the conventional GA is too slow to be used in a realistic scheduling due to its time-consuming iteration. This paper presents an improved genetic algorithm for scheduling independent tasks in grid environment, which can increase search efficiency with limited number of iteration by improving the evolutionary process while meeting a feasible result.

1. Introduction

With the rapid development of high-speed networks and the faster computing capacity of computers, a computational grid makes it feasible to provide more pervasive and cost-effective access to a collection of distributed computer resources [1]. However, it is a challenging problem to organize these resources to meet the requirements of the large scale applications of the grid. The grid environment is dynamic, in other words, the numbers of resources and tasks to be scheduled are usually variable. This kind of characteristics of grid makes the scheduling approach a complex optimization problem. Genetic algorithm (GA) has been widely used to solve these difficult NP-complete problems, as reported in [2, 3, 4, 5], whereas they ignored how to speed up convergence and shorten the search time of GA. The conventional GA is too slow to be used in a realistic scheduling due to its time-consuming iteration.

On the other hand, in Grid environment, users with common interests will be organized as virtual organization [6]. They always submit similar tasks onto geographically distributed computing resources. For example, Common Instrument Middleware Architecture (CIMA) Crystallography portal [7] provides access to X-ray crystallography data obtained from instruments and sensors. Crystallographer can submit a task on remote machine (processed by SAINT [8], an application provided by Bruker AXS) through portal to calculate the position and intensity of diffraction spots, which is the precondition to obtain a precise 3D structure of a chemical compound. Although these tasks consist of several sub tasks, such as creating directory and transferring files remotely, they have to be dispatched to the same machine to accomplish the calculation. So they can be regarded as a single task and no communications between the tasks are needed. This paper presents an improved genetic algorithm (IGA) for scheduling independent tasks in grid environment, which can increase search efficiency with limited number of iteration by improving the evolutionary process while meeting a feasible result.

The remainder of this paper is structured as follows: section 2 explicates the improved genetic algorithm; section 3 is concerned with the description of computer simulation; section 4 gives concluding remarks.

2. An Improved Genetic Algorithm for Grid Scheduling

Genetic algorithm may be used to solve optimization problems by mimicking the genetic process of biological organisms [9]. A potential solution to a specific problem may be represented as chromosome containing a series of genes. A set of chromosomes make up of a population. Each chromosome is referred to as an individual in the population. By using selector, crossover and mutation operators, GA is able to evolve the population to generate an optimal solution. This paper gives an improved GA to speed up the convergence and shorten the search time, which is on

The Sixth International Conference on Grid and Cooperative Computing(GCC 2007)
0-7695-2871-6/07 $25.00 © 2007 IEEE

the basis of an assumption that there are sufficient arriving tasks in order to make GA suitable for the scheduling algorithm.

Let N be the total number of tasks to be scheduled and W_i, $i= 0,1,2,...,N-1$ be the workload of each task in number of cycles. The workload of tasks may be obtained from historical data, such as the data size of a waiting task. Let M be the total number of computing resources and CP_j, $j=0,1,2,...,M-1$ be the computing capacity of each resource expressed in number of cycles per unit time. The generic services provided by Globus Toolkit [10], such as Monitoring and Discovery Service (MDS), make it easy to discover and maintain resource information, which is needed by scheduling algorithms to dispatch tasks onto computing resources.

The expected execution time EET_{ij} of task T_i on resource R_j is defined in formula 1. The workload W_i of task T_i is divisor and the computing capacity of resource R_j is dividend. EET_{ij} is equal to the quotient.

$$EET_{ij} = \frac{W_i}{CP_j} \qquad \textit{Formula (1)}$$

2.1 Chromosome Presentation

The efficiency of GA depends largely on the presentation of a chromosome which is composed of a series of genes. In this paper, each gene represents a task and the corresponding value of the gene represents the index of a resource onto which the task is assigned. The length of a chromosome equals N with the index range from 0 to (N-1) and the value of a gene within the range of 0 to (M-1). The following is the presentation schema:

Chromosome=($gene_0$,$gene_1$,…,$gene_{N-1}$)

0, M-1, … , 3 value of gene

↓ ↓ ↓

R_0, R_{M-1}, …, R_3 resource R_j

2.2 Population Initialization

The size of the population (*popSize*) is set to 50, so there will be enough potential solutions in a population. The individuals of the initial population are generated randomly. The allowed maximum of evolution times is set to 300. These parameters can be adjusted as needed.

2.3 Fitness Function

A fitness function must be devised to determine the fitness of a given chromosome instance. It always returns a single numerical value. The higher the return value, the better the instance.

Makespan, namely the minimum completion time of the last finishing task, is the general primary objective in performance measure of scheduling problems. In order to formulize the fitness function there are several hypotheses to simplify the constraints:

- One computing resource can only process one task at a time
- When a resource completes a task, it continues to process the next one.

The fitness function is expressed below:

$$C_m = \sum_n EET_{nm}, m = 0,1,2,...,M-1$$

$$makespan = Max\{C_m\} \qquad \textit{Formula (2)}$$

$$fitness = \frac{10000}{\alpha * L_m + \beta * makespan}$$

C_m is the sum of EET of each task T_n assigned to resource R_m, which approximately denotes the completion time of the last task on resource R_m. L_m stands for the total number of tasks assigned on resource R_m, which is used to calculate the value of fitness, consequently the load balance among computing resources is also taken into consideration. The reciprocal of makespan is selected as part of the fitness value, so that the bigger fitness value represents the better solution. α and β are set to 0.5 respectively which are decided based on the simulation results.

2.4 Evolutionary Process

Evolutionary process is accomplished by applying selection, crossover and mutation operators from one generation to the next. Selection operator determines how many and which individuals will be kept in the next generation; crossover operator controls how to exchange genes between individuals; mutation operator allows for random gene alteration of an individual. Besides the conventional genetic operators, the most important actions of the algorithm are that a verification phase is added to determine whether the evolution reaches the termination criteria and a shrink phase to control the scope of candidates. The evolutionary process is shown in figure 1.

0-7695-2871-6/07 $25.00 © 2007 IEEE

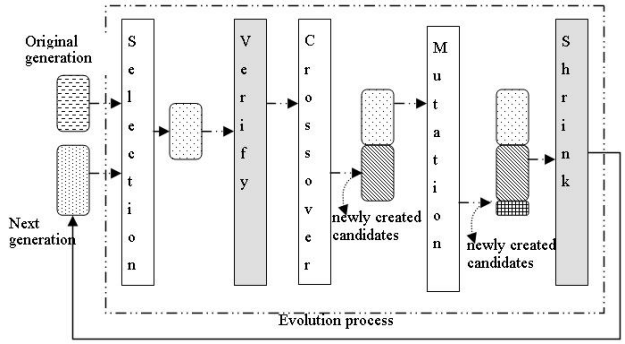

Figure 1. Evolutionary process

Firstly, the relative fitness (shown in formula 3) and cumulative proportion of each chromosome are calculated so as to carry out the roulette wheel selection [11] with a constant population size (*popSize*=50). At the same time, the best chromosome is remained for the next generation so that the algorithm always converges to the global optimum [12].

$$p_i = \frac{f_i}{\sum_{k=0}^{curSize-1} f_k} \qquad \textit{Formula (3)}$$

Selection operator procedure
BEGIN
(1) calculate the relative fitness of each chromosome in Formula 3
(2) calculate the cumulative proportion, curSize is the number of candidates
  for i=curSize-2 downto 0 do
    $P_i = P_i + P_{i+1}$
  end do
(3) remain the best chromosome in the next generation
(4) the rest of chromosomes are selected while keeping the population size constant
  for i=1 to popSize-1 do
    roulette wheel selection according to the cumulative proportion
  end do
END

Secondly, the termination condition is verified. The standard deviation of fitness value shown in formula 4 is used as the stopping criteria. When *sd* is less than a given threshold ξ the evolution is terminated, otherwise continue the iteration.

$$sd = \sqrt{\frac{\sum_{i=0}^{popSize-1} \left(f_i - \overline{f}\right)^2}{popSize}} < \xi \qquad \textit{Formula (4)}$$

$\overline{f}$ represents the average fitness of all chromosomes generated from selection phase. Threshold ξ is equal to 0.1 which is decided empirically.

Then, if the algorithm doesn't reach the termination condition, one-point crossover operator is applied to the chromosomes from selection phase. Crossover operator arbitrarily selects two chromosomes, picks a random locus (gene location), and then swaps that gene and all genes to the right of that gene between the two chromosomes. It keeps both the original and newly produced chromosomes as candidates. The number of iterations is equal to (*popSize*/2).

Crossover operator procedure
BEGIN
  for i=1 to popSize/2 do
   (1) randomly select two chromosomes
   (2) randomly select a locus to do crossover
   (3) for j=locus to (N-1) do
       swap the genes
     end do
   (4) add the newly produced chromosomes as candidates
  end do
END

After that, uniform mutation operator [13] is also applied to the chromosomes from selection phase. Mutation operator runs through the genes in each of the chromosomes and mutates each gene according to an adaptive mutation rate P_m in formula 5. C is equal to 0.01, f denotes the fitness value of the selected chromosome, f_{max} and f_{min} denote the maximum and minimum fitness value of the current generation respectively excluding those candidates produced by crossover operator. When the chromosome is near to the current optimal solution, the mutation rate is lowered, otherwise it is increased. P_m ranges from 0.1 to 0.001. The mutated chromosomes will also be kept as candidates. The number of iterations is equal to *popSize*.

$$p_m = C * \frac{f_{\max} - f}{f - f_{\min}}, \left(0.0001 \le p_m \le 0.1\right) \qquad \textit{Formula (5)}$$

Mutation operator procedure
BEGIN
  for i=1 to popSize do
   (1) calculate the adaptive mutation rate P_m in formula 5
   (2) for j=0 to (N-1) do
     /*N is the number of genes*/
     mutate each gene according to P_m
    end do
   (3) add the modified chromosome as candidate

end do
END

Finally, all chromosomes, including candidates generated by crossover operator and mutation operator, are sorted as a descending chain based on their fitness value. Then poor chromosomes are removed from the end of the chain which has lower fitness value. The purpose of this phase is to remain the best chromosome and to shrink the scope of chromosomes so as to keep a fixed size of chromosomes for selection phase. On the one hand, if the size of chromosomes is too small, it will affect the quality of the optimal solution; on the other hand, if it is too large, it will have a negative impact on the speed of convergence. So the size is decided empirically and ranges from 0.6 to 0.65. When the algorithm converges to the global optimum, the disparities among the chromosomes in the current generation are narrowed as shown in formula 6, where t represents the index of generations; f_{max} denotes the best solution in the current generation and f_{min} the worst. Consequently, the standard deviation of fitness value mentioned above can be used as the stopping criteria. After this phase, the algorithm continues to the next iteration.

$$\because f_{\max}(t+1) = f_{\max}(t), f_{\min}(t+1) \geq f_{\min}(t)$$
$$\therefore \lim_{t->\infty} f_{\min} = f_{\max}$$

Formula (6)

Shrink procedure
BEGIN
(1) sorting all chromosomes as a descending chain according to fitness value
(2) set maxSize to popSize*(1+K), K ranges from 0.6 to 0.65
(3) set curSize to the number of all chromosomes
(4) while (curSize > maxSize) do
(4.1) remove a chromosome from the end of the sorting chain
(4.2) set curSize to (curSize-1);
end do
END

2.5 Performance Analysis

In this way the algorithm only needs to execute a limited iteration to come up with an optimal solution. However, the shortcoming that is the algorithm may converge to a local optimum. So the crossover and mutation operator and a moderate fixed size of chromosomes in shrink phase as stated above are used to maintain the diversity of the evolved population and keep it from getting stuck in local optima. Although the size of population is increased by applying the crossover and mutation operator to enhance the global search capability, while at the same time, the shrink phase is used to sort the individuals according to their fitness value and remove those with lower fitness value to keep a fixed population size, which is an implicit selection accompanying with the natural selection to decrease the standard deviation of the fitness value among individuals. So it can reach the stop criteria and shorten the search time.

According to the simulation results, it is proved that our improved genetic algorithm is effective to speed up convergence while meeting a feasible result.

3. Simulation Results

Java Genetic Algorithms Package (JGAP 3.0) [14] is a set of genetic algorithms components written in Java which provide basic genetic mechanisms that can be easily used to apply evolutionary principles to problem solutions. Due to the simple and platform-independent characteristics of Java language, JGAP was adopted as the foundation to implement our improved genetic algorithm (IGA) and tests were run on a machine with Intel P4 2.80GHz CPU and 512M memory on Windows XP operating system. The simulation results of IGA are compared with those of conventional GA and Min-min heuristics [15]. The pertinent parameters of conventional GA are listed in table 1:

Table 1: Parameters of conventional GA

Parameter	Value
Maximum number of iteration	300
Size of population	50
Proportion of crossover, P_c	0.8
Proportion of mutation, P_m	0.01

The simulation results are shown in table 2. The sample data are divided into three groups with each group being experimented 50 times. In group A, there are 5 resources and 20 tasks; in group B, there are 5 resources and 40 tasks; in group C, there are 8 resources and 60 tasks. The reason of testing grouping is to represent the dynamic behavior of Grid environment. The computing capacity of resources ranges from 2 to 8, which simulates the feature of diversity of resources, and the workload of tasks ranges from 100 to 150, which imitates similar time-consuming tasks. In table 2, *IGA* denotes the result when IGA reaches the stopping criteria; *GA* denotes the result when conventional GA finishes the maximum iteration; *Min-min* denotes the value of makepsan of Min-min heuristics, which is compared to the results of IGA. Each entry of *IGA* and *GA* consists of four values, the generation index, the fitness value, the corresponding makespan and consumed time, for

example, 32:186.92:105:0.09 means that the generation index is 32, the fitness value is 186.92, the relevant makespan is 105 and the time the algorithm consumed is 0.09 seconds. In table 2, the values of average fitness of *IGA* and *GA* are also provided as well as the values of average makespan of *IGA*, which are used for the purpose of comparison.

Figure 2 shows the lines with markers displayed at each fitness value of *IGA* and *GA* for group A, figure 3 for group B and figure 4 for group C. By comparing the results of *IGA* and *GA* in each group, our improved GA is proved to be an effective way to enhance the search performance for genetic algorithm. The time consumed by *GA* is almost 2 to 4 times as much as that of *IGA,* while the scheduling solution produced by *IGA* is feasible and close to that of *GA* by comparing the value of makespan. Furthermore, the solution provided by *IGA* is much better than Min-min's, especially with the increase of the number of tasks.

The simulation result is consistent with the performance analysis in Section 2.5, which clarifies that the improvement to the evolutionary process is reasonable and effective. Moreover, the result obtained from *IGA* is feasible in terms of makespan. So it is suitable to be used in a realistic scheduling environment.

Table 2. Simulation results

	A (5 resources, 20 tasks)		B (5 resources, 40 tasks)		C (8 resources, 60 tasks)	
	IGA	GA	IGA	GA	IGA	GA
1	32:186.92:105:0.09	300:190.48:102:0.28	41:96.15:197:0.14	300:97.56:194:0.55	63:78.74:245:0.36	300:86.58:227:0.98
2	24:183.49:106:0.08	300:186.92:105:0.27	142:96.15:197:0.45	300:98.04:195:0.59	92:86.96:224:0.44	300:87.72:222:1.02
3	63:185.19:106:0.13	300:186.92:105:0.28	46:94.34:201:0.17	300:97.09:195:0.55	81:87.72:223:0.41	300:87.72:224:1.00
4	28:185.19:105:0.09	300:185.19:106:0.27	55:96.62:196:0.19	300:97.09:195:0.55	64:83.68:233:0.33	300:88.50:219:1.02
5	32:186.92:105:0.09	300:188.68:104:0.28	81:98.04:195:0.28	300:98.04:193:0.58	43:78.74:247:0.23	300:87.72:222:1.02
6	62:188.68:104:0.14	300:188.68:104:0.27	41:96.15:197:0.16	300:98.04:197:0.58	52:87.34:223:0.31	300:89.29:218:1.03
7	19:183.49:107:0.08	300:188.68:104:0.28	60:95.69:198:0.19	300:97.56:194:0.58	56:86.58:227:0.30	300:87.72:222:1.00
8	15:186.92:104:0.06	300:188.92:104:0.27	68:97.09:195:0.23	300:99.01:195:0.56	54:77.22:250:0.28	300:87.72:224:1.03
9	26:181.82:107:0.08	300:188.68:104:0.27	93:95.69:198:0.31	300:97.09:195:0.58	79:86.21:223:0.39	300:87.34:225:0.98
10	39:186.92:103:0.11	300:186.92:104:0.28	45:94.34:201:0.17	300:96.62:196:0.58	82:87.34:225:0.44	300:87.34:225:0.97
11	123:188.68:104:0.22	300:190.48:103:0.28	118:95.69:205:0.36	300:96.15:197:0.56	50:85.11:227:0.29	300:87.72:222:1.03
12	41:186.92:104:0.11	300:188.68:104:0.28	57:98.04:193:0.20	300:97.56:198:0.56	43:86.58:225:0.25	300:86.96:224:1.02
13	58:188.68:104:0.13	300:190.48:103:0.28	65:94.79:200:0.22	300:97.56:194:0.56	55:79.05:240:0.28	300:88.89:219:0.99
14	14:180.18:107:0.06	300:186.92:104:0.27	45:95.69:198:0.17	300:98.04:193:0.55	57:75.76:257:0.31	300:87.34:225:0.98
15	30:183.49:107:0.09	300:188.68:104:0.28	67:96.15:197:0.24	300:98.04:196:0.58	61:86.96:226:0.31	300:87.34:225:1.03
16	88:188.68:104:0.17	300:188.68:104:0.28	36:93.46:210:0.14	300:96.15:193:0.56	61:79.68:242:0.34	300:88.50:222:0.98
17	10:181.82:106:0.05	300:185.19:106:0.28	59:94.79:196:0.20	300:97.56:194:0.58	78:87.72:224:0.41	300:86.96:226:0.98
18	167:190.48:102:0.25	300:185.19:106:0.28	72:96.62:196:0.25	300:98.52:194:0.59	70:86.58:225:0.34	300:88.50:221:1.02
19	18:185.19:104:0.08	300:188.68:103:0.28	32:95.24:199:0.14	300:97.56:194:0.56	51:86.58:223:0.28	300:87.34:223:0.98
20	36:185.19:105:0.09	300:186.92:104:0.27	45:94.79:200:0.16	300:97.56:194:0.58	88:81.97:238:0.47	300:86.21:226:1.03
21	18:185.19:105:0.08	300:190.48:103:0.30	37:96.15:197:0.14	300:96.15:197:0.56	68:84.03:232:0.34	300:86.58:227:0.98
22	73:186.92:104:0.16	300:188.68:103:0.28	37:96.15:201:0.16	300:98.04:196:0.58	32:79.37:243:0.20	300:88.50:221:0.98
23	48:183.49:106:0.13	300:183.49:107:0.28	59:97.56:192:0.19	300:97.09:196:0.56	123:86.96:226:0.56	300:88.11:221:1.00
24	36:183.49:107:0.11	300:185.19:105:0.30	62:98.04:194:0.20	300:99.01:195:0.59	72:87.34:221:0.38	300:87.34:225:1.00
25	35:188.68:103:0.09	300:188.68:104:0.28	124:98.04:197:0.39	300:98.04:196:0.56	55:85.47:228:0.28	300:86.96:222:1.03
26	58:183.49:107:0.13	300:185.19:106:0.28	45:94.79:200:0.17	300:96.62:197:0.58	60:86.21:226:0.31	300:86.58:227:1.02

0-7695-2871-6/07 $25.00 © 2007 IEEE

27	113:185.19:105:0.20	300:190.48:102:0.27	138:96.15:197:0.44	300:96.15:199:0.56	84:86.96:224:0.38	300:86.96:226:0.99
28	90:185.19:106:0.17	300:188.68:103:0.28	80:96.62:195:0.27	300:97.56:198:0.58	56:84.03:234:0.33	300:88.50:222:0.99
29	38:186.92:104:0.13	300:186.92:105:0.28	57:95.24:199:0.22	300:97.56:198:0.56	49:85.84:229:0.27	300:88.11:221:1.05
30	38:186.92:104:0.09	300:190.48:103:0.27	69:97.09:195:0.23	300:98.04:197:0.59	75:85.84:227:0.38	300:88.89:221:1.00
31	95:190.48:103:0.19	300:190.48:102:0.30	46:97.09:196:0.17	300:98.04:197:0.59	87:85.84:229:0.42	300:87.34:223:1.02
32	43:186.92:104:0.11	300:192.31:101:0.28	29:94.34:197:0.11	300:98.52:196:0.61	57:84.03:234:0.30	300:88.11:222:0.99
33	169:186.92:103:0.18	300:188.68:104:0.27	40:95.69:198:0.16	300:97.56:194:0.58	63:86.96:224:0.33	300:86.96:226:1.05
34	27:188.68:104:0.08	300:186.92:105:0.27	57:96.62:196:0.19	300:97.56:194:0.56	105:86.58:225:0.48	300:87.34:223:1.02
35	51:186.92:105:0.11	300:186.92:105:0.31	90:97.09:195:0.28	300:99.01:193:0.59	63:83.33:236:0.31	300:87.72:224:1.00
36	43:188.68:104:0.10	300:185.19:105:0.27	61:96.15:197:0.20	300:98.04:194:0.58	55:76.34:253:0.28	300:88.11:222:0.99
37	25:183.49:107:0.08	300:188.68:104:0.28	54:97.09:195:0.19	300:98.04:197:0.59	65:85.84:227:0.34	300:87.34:223:1.03
38	62:186.92:104:0.13	300:186.92:103:0.28	28:93.02:209:0.14	300:97.56:198:0.56	39:84.75:230:0.22	300:86.96:223:1.03
39	37:190.48:103:0.11	300:188.68:104:0.28	48:94.34:201:0.17	300:98.52:193:0.56	65:87.34:224:0.36	300:87.72:222:1.00
40	33:185.19:105:0.11	300:190.48:102:0.28	64:94.79:207:0.22	300:97.56:195:0.58	59:86.21:228:0.27	300:87.72:224:1.02
41	14:186.92:105:0.06	300:188.68:103:0.28	55:93.90:202:0.19	300:95.69:194:0.58	53:86.21:223:0.28	300:87.72:223:1.03
42	13:180.18:109:0.08	300:188.68:103:0.27	38:96.15:197:0.13	300:97.56:197:0.59	62:87.34:225:0.34	300:88.11:221:1.02
43	45:185.19:104:0.13	300:186.92:104:0.28	40:95.24:199:0.16	300:97.09:195:0.56	120:85.11:231:0.55	300:87.34:225:0.98
44	100:188.68:103:0.17	300:188.68:104:0.28	68:98.04:196:0.22	300:98.04:194:0.56	62:76.34:248:0.25	300:87.72:223:1.02
45	39:186.92:104:0.11	300:188.68:102:0.28	37:96.62:196:0.13	300:97.56:194:0.58	57:86.21:224:0.28	300:88.50:218:1.06
46	12:183.49:107:0.05	300:190.48:102:0.28	57:94.79:200:0.19	300:95.69:205:0.58	77:81.63:232:0.36	300:86.96:226:1.05
47	56:188.68:103:0.14	300:192.31:101:0.27	40:94.79:200:0.16	300:96.62:198:0.58	68:86.96:224:0.34	300:88.11:221:1.09
48	72:186.92:105:0.16	300:185.19:105:0.27	49:97.56:198:0.17	300:98.04:194:0.56	104:88.50:221:0.45	300:87.34:225:1.03
49	17:181.82:108:0.08	300:186.92:103:0.28	47:97.09:195:0.17	300:98.04:197:0.58	105:86.58:220:0.53	300:88.89:221:1.00
50	24:183.49:105:0.09	300:190.48:102:0.25	87:95.69:198:0.28	300:97.56:198:0.58	57:86.58:225:0.28	300:87.34:223:1.03
Ave fitness	185.95	188.21	95.95	97.55	84.55	87.67
IGA Ave makspan		makspan =104.9	makspan = 198.2		makspan =230.4	
Min_min	makspan = 158		makspan = 393		makspan =402	

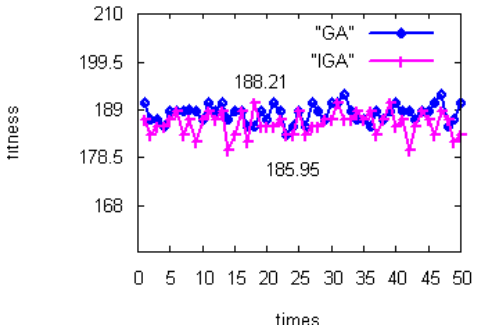

Figure 2. Simulation result of group A

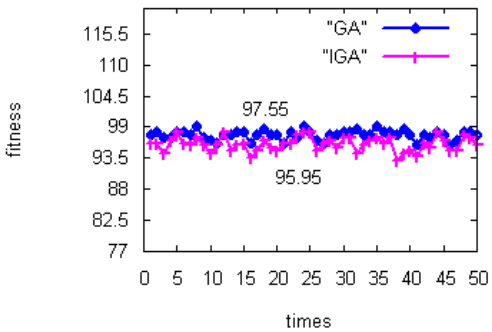

Figure 3. Simulation result of group B

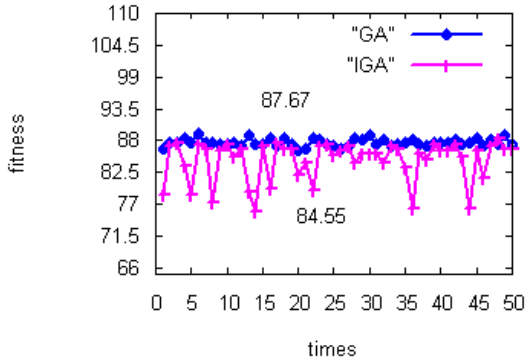

Figure 4. Simulation result of group C

4. Conclusion

This paper gives an improved genetic algorithm with limited number of iteration to schedule the independent tasks onto Grid computing resources. The evolutionary process is modified to speed up convergence as a result of shortening the search time, at the same time obtaining a feasible scheduling solution. According to the simulation results, our algorithm has better result than Min-min heuristics and better search performance than conventional genetic algorithms. Above all, the limited iteration with a feasible result makes genetic algorithm suitable for realistic scheduling in Grid environment.

Reference

[1] I. Foster, C. Kesselman, "The Grid: Blueprint for a Future Computing Infrastructure", Morgan Kaufmann Publishers 1998.

[2] V. D. Martino, M. Mililotti, "Scheduling in a Grid Computing Environment using Genetic Algorithm", Proceedings of the 16th International Parallel and Distributed Processing Symposium, p.297, April 15-19, 2002.

[3] J. Carretero, F. Xhafa, "Use of Genetic Algorithms for Scheduling Jobs in Large Scale Grid Applications", ŪKIO TECHNOLOGINIS IR EKONOMINIS VYSTYMAS, 2006 Vol XII No 1 11-17

[4] V. D. Martino, "Sub Optimal Scheduling in a Grid Using Genetic Algorithms," *ipdps*, p.148a, International Parallel and Distributed Processing Symposium (IPDPS'03), 2003

[5] Y. Gao, H. Rong and J. Z. Huang, "Adaptive grid job scheduling with genetic algorithms", *Future Generation Computer Systems,* Volume 21, Issue 1, 1 January 2005, Pages 151-161, doi:10.1016/j.future.2004.09.033

[6] I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", Intl. J. of Supercomputer Applications, 2001, 15(3)

[7] H. Yin, D. McMullen, M. Nacar, M. Pierce, K. Huffman G. Fox and Y. Ma, "Providing Portlet-Based Client Access to CIMA-Enabled Crystallographic Instruments, Sensors, and Data", proceedings of the 7th IEEE/ACM International Conference on Grid Computing (Grid 2006).

[8] SAINT, http://xray.utmb.edu/saint.html

[9] DE Goldberg, "Genetic Algorithm in Search, Optimization and Machine Learning", Addison-Wesley Publishing Company, Inc., 1989

[10] I. Foster, C. Kesselman, "Globus:A Metacomputing Infrastructure Toolkit", Intl J. Supercomputer Applications, 1997, 11(2):115.

[11] KA De Jong, "An Analysis of the Behavior of a Class of Genetic Adaptive Systems", Ph.D. thesis, University of Michigan, No.76-9381, 1975

[12] G. Rudolph, "Convergence Analysis of Canonical Genetic Algorithm", IEEE Transactions on Neural Network, 1994, 5(1):96

[13] Michalewicz Z., Janidow C., Brawezyd J., "A Modified Genetic Algorighm for Optimal Control Problems", Computer Math Application, 1992, 23(12):83

[14] K. Meffert, A. Meskauskas, N. Rotstan, J. Vos, Java Genetic Algorithms Package Project, http://jgap.sourceforge.net

[15] M. Maheswaran, "Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems", Journal of Parallel and Distributed Computing 59, 107_131 (1999)

0-7695-2871-6/07 $25.00 © 2007 IEEE