Practical Task Flow Scheduling for High Throughput Computational Grid

Wei Sun [1], Yuanyuan Zhang [1], Yanwei Wu[3], and Yasushi Inoguchi [2]

[1]Graduate School of Information Science,
[2]Center for Information Science,
Japan Advanced Institute of Science and Technology,
1-1, Asahidai, Nomi, Ishikawa, 923-1292 Japan
{sun-wei, inoguchi, yuanyuan}@jaist.ac.jp

[3]Department of Computer Science,
Illinois Institute of Technology,
10 W. 31st Street, Chicago
ywu24@cs.iit.edu

Abstract

In a practical computational grid system, task scheduling in local resource management normally is affected by the arrival rate of tasks and the sizes of tasks, that is, the scheduler must deal with the dynamic task flow. On the long-term viewpoint it is necessary and possible to improve the performance of the scheduler serving the dynamic task flow. In this paper we developed a scheduling strategy which adapts to the dynamic task flow and a genetic algorithm which balances the loads of the nodes furthest. We simulated task flows with several arrival rates and average sizes of tasks, the scheduler with our strategy and algorithm, and the schedulers with other strategies and algorithms. The simulation results show that our scheduler can adapt to the change of arrival rates better than other schedulers.

1. Introduction

The computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities [1]. The key to achieving this purpose is highly efficient resource management. The main job of resource management is to allocate tasks received from upper level to resources within its precinct. Normally this is a problem of task scheduling.

Task scheduling for the computational grid is continuous and dynamic. However, most task scheduling strategies and algorithms are affected by the tidal task flow. It is really necessary and possible to perform a long-term optimization on schedulers. We propose a methodology for task scheduling which can achieve the highest possible throughput and utilization in view of the arrival rate of tasks and task sizes.

The organization of this paper is as follows. In Section 2 the abstract model of task scheduling in computational grid and the problem of scheduling task flow are introduced. The scheduling strategy is described in Section 3. In Section 4 a genetic algorithm is presented. We made a simulation and performed some experiments, which are documented in Section 5 along with the results. Section 6 reviews some related work. Section 7 concludes this article.

[*]This research is conducted as a program for the "21st Century COE Program" by Ministry of Education, Culture, Sports, Science and Technology, Japan.

2. Model and problem

In a computational grid, resources are shared by many users, who submit their applications concurrently. Since tasks come from many unrelated users, it is feasible to assume tasks to be independent. The resource management usually receives tasks with different computation sizes at variant arrival rates. One of the objectives of resource management is to allocate the tasks to the set of computational nodes. We present an abstract model of this kind of task scheduling in Figure. 1.

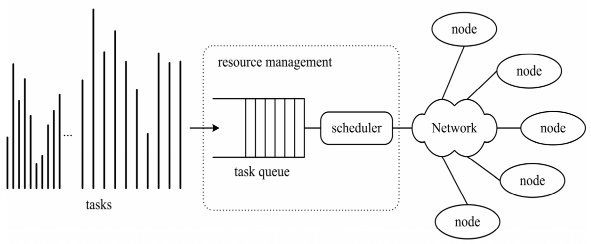

Figure 1. Model of task scheduling in the resource management of computational grid. The black lines on the left denote tasks. The intervals between the lines indicate the intervals of task arrival. The length of each line denotes the task size.

When the arrival rate of tasks is high enough, the scheduler can always collect enough tasks and the computational nodes are always busy. The throughput and the utilization reach the maximum, which are decided by the total processing ability of the system. We defined this phase of system status as *saturation.* Most researches on task scheduling focused on the saturation phase to optimize scheduling algorithms to shorten the maxspan. The maxspan is basically the largest task completion time among all the nodes in the system [2].

When the arrival rate of tasks is low, the scheduler might be idle and the computational nodes could be left unused. The throughput and the utilization are affected mainly by the arrival rate of tasks and the task sizes. We defined this phase of system status as *starvation.* The optimization on task scheduling can be of little help in the deep starvation phase.

For any definite system, when the supply of computation, i.e. the total processing capacity, can just satisfy the requirement of computation from users, the system status is on the boundary between the saturation phase and the starvation phase. We called this boundary as *balance line*. Around the balance line there is a region where the coming low arrival rate and the coming small average task size do not move the system status into the starvation phase at once or vice versa. This is due to the historical light load or heavy load on the nodes. The system status in this region is easy to slide into the deep saturation phase or the deep starvation phase. We named this region as slide region. Figure 2 illustrates the relationship of the arrival rate, task size, and the system status.

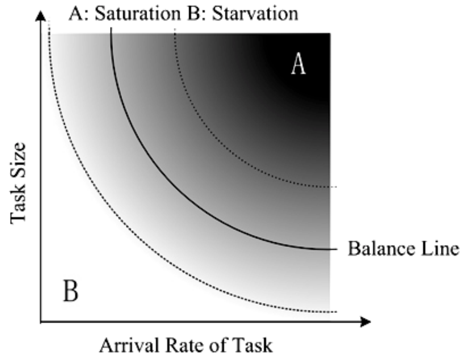

Figure 2. Two phases of system status. The darker the color is, the busier system is. The slide region is marked with two dot lines.

A scheduler normally consists of the scheduling algorithm and the scheduling strategy. The scheduling algorithm tries to map one task or a batch of tasks onto the computational nodes, where the workloads are balanced as possible. In theory the more balanced the workloads are, the shorter the maxspan is and the more efficient the system is. There are two main classes of scheduling algorithm: the immediate mode scheduling and the batch mode scheduling. It is commonly believed that the batch mode can lead to a shorter maxspan than the immediate mode under the precondition that the scheduler can collect enough tasks, but there is no guarantee for collecting enough tasks for the scheduler serving a dynamic task flow. On the other side the immediate mode can approach the same result of the batch mode or better when the number of tasks is quite small [3]. The scheduling strategy is used to control the scheduling algorithm to create schedules. The scheduling strategy with simple timer or counter can not adapt the scheduler to the dynamic task flow. We are going to present a task scheduling method which can perform well and adapt to the variety of task flows.

3. Scheduling strategy

Generally the task flow scheduler fulfills a scheduling process after every time interval, called the *scheduling cycle*. In every scheduling cycle one or a batch of tasks are allocated to computational nodes. We named our scheduling strategy as *dynamic scheduling cycle*. Some notations to appear in this paper are listed in Table 1.

Table 1: Notation list

Notation	Definition
N	Task set; tasks in the task queue
M	Set of Computational node
$\lvert\cdot\rvert$	Cardinality of set
L_i	Total size of tasks at node *i*
λ	Average arrival rate of tasks
λ_H	A high arrival rate of tasks
λ_L	A low arrival rate of tasks
C_i	Processing capacity of node *i*
s_i	Computation size of task *i*
S_E	Average size of tasks
c_{ij}	The communication cost of task *i* to be send to node *j*
r_i	The remaining execution time of the task currently being processed by the ith node
t_l	The shortest execution time of the tasks ready to be processed by nodes, $t_l = \min(\frac{L_j}{C_j} + r_j), j \in [1, \lvert M \rvert]$
t_s	time needed to create a task schedule

In our strategy a new scheduling cycle starts only if there are almost no tasks ready to be executed on the computational nodes, so that the length of the scheduling cycle is changed dynamically as the system status moves between the saturation phase and the starvation phase. Because the arrival rate has a direct impact on the number of tasks in task queue, and the shortest execution time of the tasks on all the nodes t_l can imply the system status, $|N|$ and t_l are used as the main parameters in the dynamic scheduling cycle strategy.

If there are enough tasks in task queue at the beginning of scheduling cycle, we chose the batch mode task scheduling algorithm in order to obtain high throughput and utilization. Considering higher and higher bandwidth, the execution time of a grid task is generally longer than the transmission time over networks. Moreover, in the saturation phase the task execution happens in parallel with the transmission of other subsequent tasks. Therefore we ignore the communication cost in batch mode scheduling. We developed a genetic algorithm, which is presented in Section 4. After the task scheduling the system status has been in the deep saturation phase or will come into.

If there are a few tasks in task queue at the beginning of scheduling cycle, the scheduler immediately sends one task to the computational node where the task can be finished earliest. In this way the scheduler allocates a few of tasks to computational nodes as soon as possible and waits for next high tide of task flow. In this moment we take the communication cost into account in order to finish tasks in the shortest time. After the task scheduling the system status has been in the deep starvation phase or will come into.

```
//Scheduling strategy;
1.while(1){
2.     update $t_l$, $t_s$;
3.     if($t_l$ > $t_s$){
4.          wait 1 second;
5.     }elseif($|N|>2|M|$){//enough tasks;
6.          GA_Scheduling();
7.     }elseif($|N| == 0$){//empty task queue;
8.          wait 1 second;
9.     }else{//a few tasks;
10.        for ($i=1$; $i < |M|$; $i$++){
11.             find the node $j$ with
                    $\min(\frac{s_i + L_i}{C_j} + c_{ij} + r_j), j \in [1, |M|]$;
12.             map task $i$ to node $j$;
           }
     }
}
```

Figure 3. Scheduling Strategy.

The strategy is shown in Figure 3. In our strategy when a node finishes all tasks and is ready to receive new tasks, we call this node as the ready node and the corresponding time as the ready time. The time t_s that the scheduler takes to create a schedule should be decided in the real environment.

4. Genetic algorithm

A GA [4, 5] is a biologically inspired search method, which partially searches for a large solution space, known as population, and uses historical information to exploit the best solution from previous searches, known as generations, along with random mutations to explore new regions of the solution space. A GA basically repeats three steps: selection, crossover, and mutation. The process combined with initiation and evaluation is shown in Figure 4. According to the nature of computational grid and our scheduling strategy, we developed a genetic algorithm for our task scheduling.

```
//Procedure of Genetic Algorithm
1.   Initiate population;
2.   Evaluation
3.   While(stop criteria not met){
4.        Selection operation;
5.        Crossover operation;
6.        Mutation operation;
7.        Evaluation;
     }
8.   Output the best solution
```

Figure 4. Procedure of a basic GA.

The encoding represents a chromosome of individual, which is a schedule. A number in the chromosome is a gene, which represents the corresponding task to be allocated in the node denoted by this number. The length of chromosome n is equal to $|N|$, and the largest number of genes m is equal to $|M|$. We use ch to denote a chromosome. A chromosome is illustrated in Figure 5.

task	1	2	3	4	5	...	$n-1$	n
node	2	1	9	m	2	...	7	3

Figure 5. Representation of chromosome.

The fitness function creates a fitness value for each individual, which indicates the quality of the scheduling. For a task scheduling problem, the ideal result is the absolute balanced workloads. Any scheduling result can only be close to the ideal result but never to reach it. We use relative error as the fitness value. The smaller fitness value implies the more balanced workloads. The ideal scheduling result is

$$t_{ideal} = \frac{\sum_{i=1}^{|N|} s_{ch[i]} + \sum_{j=1}^{|M|} L_j + \sum_{l=1}^{|M|} (r_l \cdot C_l)}{\sum_{l=1}^{|M|} C_l} .$$

The time needed to finish a batch of tasks on a node is

$$t_{node_k} = \frac{\sum_{ch[p]=k} s_p + L_k}{C_k} + r_k .$$

Thus our fitness function is

$$Fit = \sqrt{\sum_{k=1}^{|M|} \left| t_{ideal} - t_{node_k} \right|^2} .$$

The initiation of population has a straightforward effect on the convergence time of the GA and the quality of the result. Our initiation was designed to guide GA to search more effective solution space by avoiding impossible allocation of task. The details of initiation are shown in Figure 6.

```
Initiation() //Initiation Algortihm;
Input: task set N, population set P; //
Output: chromosome ch_1, ch_2, ch_3, … ch_p
{
1.    for (i=1; i<= |P|; i++){
2.          Initiate_chromosome (N, ch_i);
      }
}
Initiate_chromosome ()//Sub-function;
Input: task set Ω;
Output: chromosome ch;
{
3.    while(Ω is not empty){
4.          select a subset ω from Ω randomly;
5.          find the task_i in ω with
              $\min(\frac{s_i + \sum_{ch[j]=k} s_j + L_k}{C_k} + r_k), k \in [1, |M|]$;
6.          ch[i] = k;
7.          remove task_i from Ω;
      }
}
```

Figure 6. Initiation Operation.

We use a tournament selection in our GA. Firstly a subset of individuals are selected from the population. Secondly the individual with the smallest fitness value is selected as one parent. Two tournaments are performed and two individuals are chosen as the parents. After the selection operation we use a two-point crossover operation [6] to reproduce the child individual.

Usually a mutation operation exchanges two randomly selected genes. But here the random selection has a pitfall: if the values of the two selected genes are identical, then the mutation operation is in vain. We describe this problem with a numeric matrix shown in Figure 7. Hence we compel the mutation operation to select two genes with different values.

$$\text{nodes}\left\{\begin{array}{ccccccccc} \multicolumn{9}{c}{\overbrace{\hspace{12em}}^{\text{tasks}}} \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & \bullet & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & & \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & & \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & & \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & & \\ \bullet & & & & & & & \bullet & \\ 0 & & & & & & & & 0 \end{array}\right.$$

Figure 7. Matrix of task allocation. "1" indicates that the task is allocated on the corresponding node. Only the "1" in different rows allow to be exchanged.

It is possible to decrease the fitness value when a task is moved from the node with the longest ready time to another node. We developed a refinery algorithm to refine the individual produced by the mutation operation. The execution time of the refinery algorithm is less than *O(mn).*

```
Refinery()// The refinery algorithm
Input: an individual with chromosome ch;
Output: refined individual with chromosome ch';
{
1.    find node j with
        $\max(\frac{\sum_{ch[i]=j} s_i + L_j}{C_j} + r_j), j \in [1, |M|]$;
2.    for(all i with ch[i] = j ){
3.          for( all k with ch[k] != j){
4.                copy ch to ch';
5.                exchange ch'[i] with ch'[k];
6.                if(ch'.fit < ch.fit) return ch';
            }
      }
}
```

Figure 8. Refinery Algorithm.

After the refinery algorithm, the solution with the largest fitness value is replaced. Note that the child solution, whose structure is identical to any of the solution structures in the population, is not allowed to enter the population. This constraint is helpful for avoiding a homogeneous population.

The GA will evolve the population until the stop criterion is met. Our stop criterion is to define a boundary generation number. After that number of generations, for example 1000, if the best fitness value of every generation is invariable or oscillates in a small range, the GA stops and outputs the best solution.

5. Simulation

A simulation was built for testing and evaluation. We implemented the scheduling strategies and algorithms in this simulation. We compared our strategy and algorithm with some well known scheduling algorithms which were Max-min, Min-min, Sufferage, and the genetic algorithm described in [8], and the most familiar scheduling strategy - the regular time interval. Each algorithm was assembled with a strategy to form a scheduler. All schedulers are listed in Table 2. We denote the scheduler with our scheduling strategy and algorithm as PTFS (Practical Task Flow Scheduling).

Table 2. Scheduler List.

Scheduler	Strategy	Algorithm
PTFS	Dynamic scheduling cycle	Our GA
GA_1	Regular time interval	Our GA
GA_2	Regular time interval	GA[8]
Max-min	Regular time interval	MaxMin
Min-min	Regular time interval	MinMin
Sufferage	Regular time interval	Safferage

The task sizes in a task flow were randomly generated and two task flows，each of which has 4000 tasks, were simulated. Ten computational nodes with different processing abilities were also simulated. The scheduler and all computational nodes are connected by a network. We scaled the processing abilities of computational nodes and task sizes simply in integers. The S_E of one task flow was 5000 and the other one is 50000. In all experiments the population size is 80. The regular time interval is 20s. For any NPC problem, GA requires no more than exponential time to produce the result, if the MCL (Minimum Chromosome Length) growth rate is no more than linear [9]. The execution time t_s of creating a schedule by PTFS is estimated by the following conservative estimate equation. The error of this estimation equation is less than 10% in the worst case.

$$t_s = 40\exp(\frac{|N|}{1580}) - 40$$

When the arrival rate of tasks is high enough and swings within a sensitive scope λ_H~λ_L, the maxspan of the task flow with S_E = 50000 is shown in Figure 9. Here the high arrival rate can keep system status always in the moderate saturation phase, and the variant arrival rate can keep system status moving between the deep saturation phase and around the balance line.

In fact most modern scheduling algorithms can be quite close to the ideal result t_{ideal}. The extent of the real result being close to t_{ideal} depends on the real problem such as the task heterogeneity and the processing capacity heterogeneity. Roughly the optimization on algorithm is limited. Our task scheduling methodology is not merely to develop an optimized genetic algorithm; furthermore the dynamic scheduling cycle strategy can make a good contribution to the scheduler as a whole. Therefore the advantage of our scheduler at the variant arrival rate is more obvious than at high arrival rate.

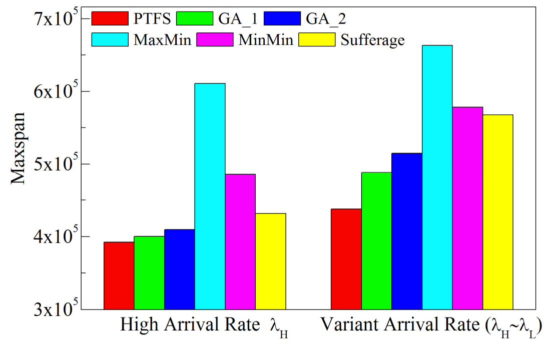

Figure 9: Maxspan (S_E = 50000).

Under the same λ_H and λ_L, the maxspan of the task flow with S_E = 5000 is shown in Figure 10.

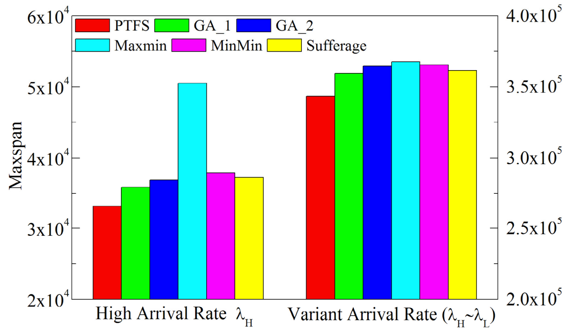

Figure 10: Maxspan (S_E = 5000).

The system status is not only related to the arrival rate of tasks but also the task size. Under the same

λ_H and λ_L the system status for $S_E = 5000$ is in the deep saturation phase and moving between the deep saturation phase and the starvation phase. If the system status is always in the starvation phase, the differences of maxspan between our scheduler and the others are small, and the maxspan in the starvation phase is mainly decided by the time that all tasks arrive at the scheduler. So we did not plot the maxspan under a quite low arrival rate.

In the saturation phase the well balanced workload implies the high resource utilization. In the starvation phase the response time of scheduler can affect the resource utilization, despite the utilization is low in the starvation phase. The regular time interval strategy can not achieve the higher utilization at the high arrival rate and the faster response time at low arrival rate than our scheduling strategy.

We define the utilization of a node simply as

$$\mu_i = 1 - \frac{\sum \text{the idle time of node i}}{Maxspan},$$

and the average utilization as

$$\mu_{average} = \frac{\sum \mu_i}{|M|}.$$

Our definition of average utilization is similar with the definition in [2], but our definition is more suitable for low arrival rate and variant arrival rate because we do not simply calculate the task completion time.

The following experiments illustrate the maxspan and the resource utilization differences between the results of PTFS and the best results of other schedulers. In these experiments the arrival rate swings around the corresponding average arrival rate with the amplitude not more than the neighboring average arrival rate. The results are shown in Figure 11 and 12. In order to plot all the data together, the maxspans are normalized and the average arrival rates of tasks are on the logarithmic scale.

The differences between PTFS and other schedulers are going to become smaller as the arrival rate of tasks becomes very small or very large. The largest differences appear in the moderate saturation phase and around the balance line. When the arrival rate of tasks reaches a high enough value, the differences between PTFS and the other schedulers will not change, and meanwhile the system status is in the utmost of the saturation phase, which is the maximum processing ability of this system. When the arrival rate is going to be quit low, the differences between PTFS and the other schedulers are going to gradually disappear, and meanwhile the maxspan and utilization are decided by the arrival of tasks.

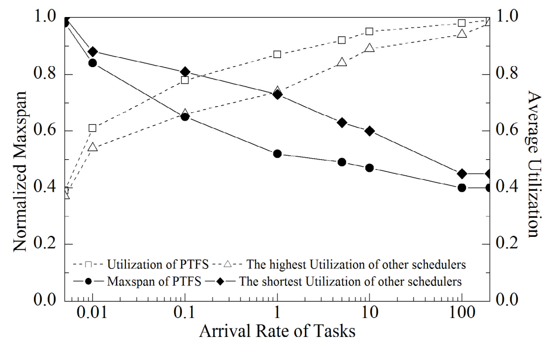

Figure 11: Utilizations and normalized maxspan at different λ (S_E = 50000).

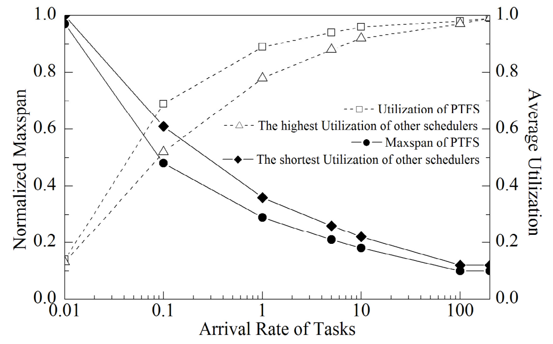

Figure 12: Utilizations and normalized maxspan at different λ (S_E = 5000).

6. Related work

Usually there is a scheduling system in a grid environment including meta-scheduler or global scheduler, global task queue, local scheduler, local task queue and so on [10, 11, 12]. Our scheduling strategy and algorithm focus on the resource level in a computational grid. The model presented in this paper aims at the local scheduling problem. The scheduling problem in grid computing environment needs the practical and realistic solutions rather than the theoretic ones. Thus it is an inevitable trend to induct the arrival of tasks and the system status into the research on task scheduling [11, 12].

The independent task mapping techniques have been well summarized and compared in [3, 7, 8]. It is shown that a genetic algorithm is an effective method for task scheduling. GA was successfully used for task scheduling in [2, 13, 14, 15, 16]. There are two main class task scheduling: the immediate mode scheduling and the batch mode scheduling. The immediate mode scheduling uses the FCFS strategy to deal with the task one bye one. For the batch mode scheduling the two basic elements which should be considered by scheduling strategy are the time and the count of tasks. The regular interval time strategy and fixed count strategy are the simplest ones [3]. In order to improve the resource utilization, a dynamic batch size strategy was used to adjust the batch size to avoid long scheduling time and idle resources [13]. A technique similar with the dynamic batch size strategy, called slide window, was used to update the number of tasks in creating the next schedule [2]. No matter whether the fixed count or the dynamic batch size, it is difficult to adapt to the dynamic task flow when the arrival rate of tasks increases or decreases, even if the dynamic batch size strategy can change the batch size according to the resource load and the execution time of scheduler.

7. Conclusion and future work

In this paper after defining the system status in terms of utilization, throughput, arrival rate of tasks and task sizes, we present a scheduling strategy to adapt to the dynamic task flow, and a genetic algorithm which is used in the scheduling strategy. According to the result of simulation our methodology works well, especially in the situations where the arrival rate of tasks swings within a scope and the mean of task sizes is large. In each scheduling cycle our methodology can achieve more or less advantage over the other schedulers, but the long-term advantage is obvious. We believe our methodology is a good solution for practical resource management in a computational grid.

In this paper the estimate equation for GA is a conservative method to calculate the execution time of our genetic algorithm. Therefore it is possible to develop a more accurate method. An alterable population size of GA is more suitable for the variant number of tasks. Thus the genetic algorithm can be improved further to shorten the convergence time. In our strategy it is possible to take the place of the genetic algorithm with another batch mode scheduling algorithm, only if the latter one can achieve better result and is easier to estimate the accurate execution time.

References

[1] Forster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, San Francisco, 1999.

[2] A. Y. Zomaya and Y. H. Teh, "Observations on Using Genetic Algorithms for Dynamic Load-balancing", *IEEE Trans. Parallel and Distributed System*, vol.12, no.9, Sep. 2001.

[3] M. Maheswaran, S. Ali, H. J. Siegel, et al., "Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing System", *Journal of Parallel and Distributed Computing*, 59, 1999, pp.107-131.

[4] J. H. Holland, *Adaptation in Natural and Artificial System*, Univ. of Michigan Press, 1975

[5] P. C. Chu, and J. E. Beasley, "A Genetic Algorithm for the Generalised Assignment Problem", *Computer & Operation Research*, 24, 1997, pp17-23.

[6] G. Syswerda, "Uniform Crossover in Genetic Algorithms", *Proc. 3rd Int'l Conf. on Genetic Algorithms*, 2-9, 1989.

[7] T. D. Braun, H. J. Siegel et al., "A Comparison Study of Static Mapping Heuristic for a Class of Meta-task on Heterogeneous Computing Systems", *Proc. 8th Heterogeneous Computing Workshop*, San Juan, Puerto Rico, 1999.

[8] T. D. Braun, H. J. Siegel et al., "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems", *Journal of Parallel and Distributed Computing*, 61, 2001, pp.810-837.

[9] B, Rylander, "Computational Complexity and the Genetic Algorithm", Ph.D. Dissertation, University of Idaho, USA, June, 2001.

[10] E. Caron, V. Garonne, A. Tsaregorodtsev, "Evaluation of Metacheduler Architectures and Task Assignment Policies for High Throughput Computing", *INRIA Technical report*, 2005, 2005-27.

[11] L. He, S. A. Jarvis, D. P. Spooner et al., "Allocating Non-Real-Time and Soft Real Time Jobs in Multiclusters", *IEEE Trans. Parallel and Distributed System*, vol.17, no.2, Feb. 2006, pp 99-112.

[12] V. Berten, J. Goossens, and E. Jeannot, "On the Distribution of Sequential Jobs in Random Broking for Heterogeneous Computational Grids", *IEEE Trans. Parallel and Distributed System*, vol.17, no.2, Feb. 2006, pp113-124.

[13] A. J. Page and T. J. Naughton, "Dynamic Task Scheduling Using Genetic Algorithms for Heterogeneous Distributed Computing", *Proc. of 19th IEEE/ACM Intl. Parallel and Distributed Processing Symposium,* Denver, Colorado, 2005.

[14] A. Y. Zomaya, C. Ward, and B. Macey, "Genetic Scheduling for Parallel Processor System: Comparative Studies and Performance Issues", *IEEE Trans. Parallel and Distributed System*, vol.10, no.8, Aug. 1999.

[15] L. Wang, H. J. Siegel, V.P. Roychowdhury, et al., "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach", *Journal of Parallel and Distributed Computing*, 1997, pp.8-22.

[16] E.S. H. Hou, N. Absari, and H. Ren, "A Genetic Algorithm for Mutiprocess Scheduling," *IEEE Trans. Parallel and Distributed System*, vol.5, no.2, 1994.