IEICE TRANS. INF. & SYST., VOL.E90–D, NO.4 APRIL 2007

PAPER

Dynamic Task Flow Scheduling for Heterogeneous Distributed Computing: Algorithm and Strategy*

Wei SUN[†a)], **Yuanyuan ZHANG**[††], ***Nonmembers***, ***and*** **Yasushi INOGUCHI**[†††], ***Member***

SUMMARY Heterogeneous distributed computing environments are well suited to meet the fast increasing computational demands. Task scheduling is very important for a heterogeneous distributed system to satisfy the large computational demands of applications. The performance of a scheduler in a heterogeneous distributed system normally has something to do with the dynamic task flow, that is, the scheduler always suffers from the heterogeneity of task sizes and the variety of task arrivals. From 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 propose a task scheduling method including a scheduling strategy which adapts to the dynamic task flow and a genetic algorithm which can achieve the short completion time of a batch of tasks. The strategy and the genetic algorithm work with each other to enhance the scheduler's efficiency and performance. We simulated a task flow with enough tasks, the scheduler with our strategy and algorithm, and the schedulers with other strategies and algorithms. We also simulated a complex scenario including the variant arrival rate of tasks and the heterogeneous computational nodes. The simulation results show that our scheduler achieves much better scheduling results than the others, in terms of the average waiting time, the average response time, and the finish time of all tasks.

key words: *heterogeneous distributed computing, task scheduling, task flow, genetic algorithm, scheduling strategy*

1. Introduction

Heterogeneous distributed computing environments utilize a distributed suite of different machines, interconnected with high-speed links, to perform different computationally intensive applications that have diverse computational requirements [1], [2]. Scheduling tasks to a set of heterogeneous machines has been shown to be NP-complete [3]. A good task scheduler is very important to exploit the true potential of a heterogeneous distributed system.

The traditional task scheduling for the heterogeneous distributed computing consists of the scheduling strategy and the scheduling algorithm. The scheduling algorithms can be classified into the immediate mode and the batch mode according to the number of tasks involved in a task schedule. The immediate mode algorithms allocate one task onto a computational node once this task arrives at the scheduler. The batch mode algorithms allocate a batch of tasks which are in the task queue of the scheduler. The scheduling strategy usually calls the scheduling algorithm to create a schedule. Sometimes the scheduling strategy is embedded implicitly in the scheduling algorithm, for example the immediate mode algorithms themselves imply that the tasks are allocated one by one, i.e., FCFS. The most familiar and simplest scheduling strategies are the regular time interval strategy, which creates a schedule every a regular time interval, and the fixed count strategy, which creates a schedule every a fixed count of tasks [4]–[6].

Task scheduling in an applied heterogeneous distributed computing system means the variant arrival rate of tasks, the high heterogeneity of tasks, and the durative scheduling process. Hence the task scheduling is continuous and dynamic. Instead of the static task scheduling, the scheduler always deals with the dynamic task flow. In view of the traditional task scheduling and the dynamic task flow, it is necessary and possible to perform a long-term optimization on schedulers. We propose a task scheduling method which consists of a scheduling strategy, which adapts to the dynamic task flow, and a genetic algorithm, which balances the loads of the nodes furthest to shorten the completion time of a batch of tasks. The performance of our scheduling method has been illustrated briefly in [7], where the task computation sizes are generated randomly and the task arrival rates are at a few discrete values. For a continuous and dynamic scheduling process the average response time of tasks is a good index to evaluate a scheduler. In this paper the research is involved with the waiting time, the response time, and the behaviors of different algorithms and strategies in a complex scenario.

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

2. Model and Problem

Normally, a heterogeneous distributed computing system consists of the local task queues on the computational nodes, the scheduler, and the scheduler task queue. We present an

Manuscript received June 28, 2006.
Manuscript revised November 20, 2006.

[†]The author is with the School of Information Science, JAIST, Nomi-shi, 923–1292 Japan.
[††]The author is with the Peta-Scale Computing Research Center, Fujitsu Laboratories Ltd., Kawasaki-shi, 211–8588 Japan.
[†††]The author is with the Center of Information Science, JAIST, Nomi-shi, 923–1292 Japan.

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

a) E-mail: sun-wei@jaist.ac.jp

DOI: 10.1093/ietisy/e90–d.4.736

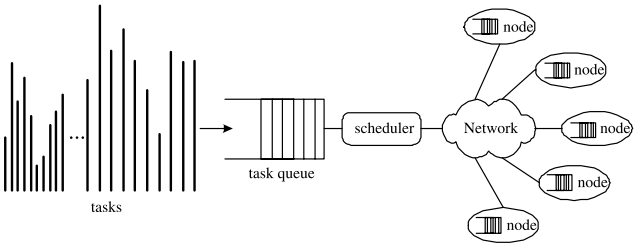

Fig. 1 Model of heterogeneous distributed computing system.

abstract model for a heterogeneous distributed computing system in Fig. 1. The black lines on the left denote tasks. The intervals between the lines indicate the intervals of task arrivals. The length of each line denotes the task size.

In the long-term scheduling process the scheduling strategy and algorithm bear different functions. The scheduling algorithm takes charge of creating schedules in which the completion time of all tasks is as short as possible. In theory the more balanced the workloads are, the shorter the total tasks completion time is and the more efficient the system is. For the two main classes of scheduling algorithms, it is commonly believed that the batch mode can lead to the shorter completion time than the immediate mode under the precondition that the scheduler can collect enough tasks, but there is no guarantee for the scheduler serving a dynamic task flow to collect enough tasks. On the other hand the immediate mode can approach the same result of the batch mode or better when the number of tasks is quite small [4]. The scheduling strategy is used to control the scheduling algorithm to create schedules. For the irregular task arrivals the scheduling strategy with the simple timer or counter can not adapt the scheduler to the dynamic task flow.

In [4]–[6], [8]–[10], it is assumed that the task computation sizes can be known or predicted before task scheduling, and the tasks are independent, nonpreemptive and do not have the real-time requirements. The model and the assumptions are valid for a heterogeneous computing system [4]–[6] or a local resource domain of Grid environment [11], [12]. Therefore all these assumptions are inherited in this paper. The difference of our work is to consider the dynamic task flow in the context of task scheduling. The motivation of our work is to achieve the smaller average response time of tasks.

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 the computational nodes. We named our scheduling strategy as dynamic scheduling cycle. Some notations to appear in this paper are listed in Table 1.

In our strategy a new scheduling cycle starts only if there are almost no tasks ready to be executed in the local task queues, so that the length of the scheduling cycle

Table 1 Notations.

Notation	Definition
N	Task set; all tasks in the scheduler task queue
M	Set of computational nodes
$\|\cdot\|$	Size of set
L_i	Total tasks size at node i
C_i	Processing capacity of node i
s_i	Computation size of task i
c_{ij}	The communication cost of task i to be sent to node j
r_i	The remaining execution time of the task currently being processed by the node i
t_l	The shortest execution time of the tasks ready to be processed by nodes t_l=min $(\frac{L_j}{C_j} + r_j)$, $j \in [1, \|M\|]$
t_s	Time needed to create a task schedule

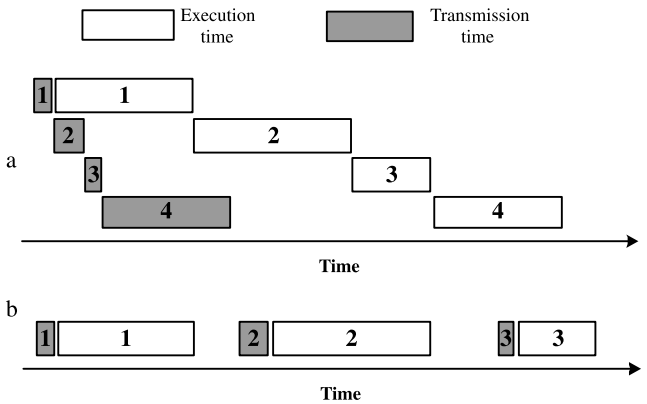

Fig. 2 The transmission time and execution time of tasks in a node. The transmission time is in parallel with the execution time, when lot of tasks are scheduled at the same time (a). In a, the transmission time of task 2, 3, and 4 does not affect their response time. The transmission time and execution time are in serial, when a few of tasks are scheduled (b). In b, the response time of each task contain its transmission time.

changes dynamically. Because the arrival rate has a direct impact on the number of tasks in the scheduler task queue, and t_l implies when all local task queues are going to be empty, $|N|$ and t_l are used to decide what time to start a new scheduling cycle and what kind of task scheduling to be adopted in the dynamic scheduling cycle strategy, instead of monitoring the arrival rate of tasks and the system load.

If there are enough tasks in the task queue at the beginning of scheduling cycle, we choose the batch mode task scheduling algorithm in order to obtain the shortest completion time. Considering higher and higher bandwidth, the execution time of a task is assumed not less than the transmission time over networks. Moreover, when some tasks are already being executed on the nodes, the execution of these tasks happens in parallel with the transmission of other subsequent tasks. In Fig. 2, the transmission time and execution time of tasks are illustrated. Therefore, we ignore the communication costs in batch mode scheduling. We developed a genetic algorithm as the batch mode scheduling algorithm, which is presented in Sect. 4.

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

Fig. 3 Dynamic scheduling cycle strategy.

```
//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 individual;
```

Fig. 4 Procedure of a basic GA.

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

Fig. 5 Representation of chromosome.

```
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 ((s_i + Σ_{ch[j]=k} s_j + L_k)/C_k + r_k), k ∈ [1, |M|];
6.   ch[i] = k;
7.   remove task_i from Ω;
   }
}
```

Fig. 6 Initiation operation.

If there are only a few tasks in the scheduler task queue at the beginning of a 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 these tasks to the computational nodes as soon as possible and waits for next high tide of task flow. In this moment, we take the communication costs into account in order to finish tasks in the shortest time, because the transmission time will affect the response time greatly which is illustrated in Fig. 2. In fact this is immediate mode scheduling.

The strategy is shown in Fig. 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, so we will introduce an expression of t_s in our simulation.

4. Genetic Algorithm

A GA [13], [14] is a biologically inspired search method, which partially searches for a large individual space, known as population, and uses historical information to exploit the best individual from previous searches, known as generations, along with random mutations to explore new regions of the population. A GA basically repeats three steps: selection, crossover, and mutation. The process combined with initiation and evaluation is shown in Fig. 4. We developed a genetic algorithm for our task scheduling method to achieve the shortest completion time of each schedule.

4.1 Encoding and Main Operations

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 number of genes m is equal to $|M|$. We use ch to denote a chromosome. $ch[i]$ means the ith gene of a chromosome. If the value of $ch[i]$ is j, it means that the ith task is scheduled to the jth node. A chromosome is illustrated in Fig. 5.

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 individual spaces by avoiding impossible task allocation. The details of initiation are shown in Fig. 6.

A tournament selection is used in our GA. First, a subset of individuals are selected from the population. Second, 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 [15] to reproduce the child individual.

Usually a mutation operation exchanges two randomly selected genes. But 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 Fig. 7. Hence we compel the mutation operation to select two genes with different values.

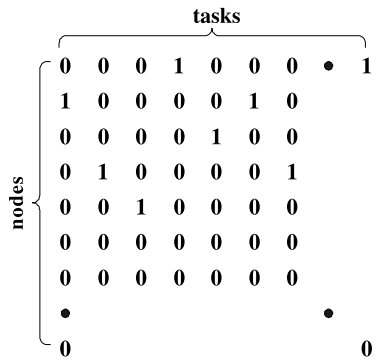

Fig. 7 Matrix of task allocation. "1" indicates that the task is allocated on the corresponding node. Only the "1" in different rows can be exchanged.

4.2 Fitness Function

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 absolutely 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 more balanced workload. The ideal ready time for all nodes is

$$t_{ideal} = \frac{\sum_{i=1}^{|N|} s_i + \sum_{j=1}^{|M|} L_j + \sum_{l=1}^{|M|}(r_l C_l)}{\sum_{l=1}^{|M|} C_l}. \tag{1}$$

The proof of Eq. (1) is as follows:

Proof. Because t_{ideal} means all workloads of computational nodes are balanced absolutely, the ready time of each node is identical to the others and also identical to t_{ideal}. The ready time is the sum of the execution time of all new arriving tasks in the local task queue, the execution time of all tasks already in the local task queue, and the remaining execution time of the task currently being processed. For each node we have the following equations and their transforms:

$$node1 : t_{ideal} = \frac{\sum_{ch[i]=1} s_i + L_1}{C_1} + r_1 \Rightarrow$$
$$C_1 t_{ideal} = \sum_{ch[i]=1} s_i + L_1 + r_1 C_1$$
$$node2 : t_{ideal} = \frac{\sum_{ch[i]=2} s_i + L_2}{C_2} + r_2 \Rightarrow$$
$$C_2 t_{ideal} = \sum_{ch[i]=2} s_i + L_2 + r_2 C_2$$
$$\vdots$$
$$node|M| : t_{ideal} = \frac{\sum_{ch[i]=|M|} s_i + L_{|M|}}{C_{|M|}} + r_{|M|} \Rightarrow$$
$$C_{|M|} t_{ideal} = \sum_{ch[i]=|M|} s_i + L_{|M|} + r_{|M|} C_{|M|}.$$

All of the $|M|$ equations are added up, so that we have the following equation:

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

Since

$$\sum_{j=1}^{|M|} \sum_{ch[i]=j} s_i = \sum_{i=1}^{|N|} s_i,$$

after moving $\sum_{l=1}^{|M|} C_l$ to the right, the proof is completed. So that Eq. (1) is the expression of t_{ideal}.

```
Refinery()// The refinery algorithm;
Input: an individual with chromosome ch;
Output: refined individual with chromosome ch';
{
1. find node j with
   max ( (Σ_{ch[i]=j} s_i + L_j)/C_j + r_j), j ∈ [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';
     }
   }
}
```

Fig. 8 Refinery algorithm.

The real ready time of a node is

$$t_{node_k} = \frac{\sum_{ch[p]=k} s_p + L_k}{C_k} + r_k. \tag{2}$$

Thus our fitness function is

$$fit = \sqrt{\sum_{k=1}^{|M|} |t_{ideal} - t_{node_k}|^2}. \tag{3}$$

4.3 Refinery Algorithm and Stop Condition

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 refinery algorithm is shown in Fig. 8. The execution time of the refinery algorithm is less than $O(mn)$.

After the refinery algorithm, the individual with the largest fitness value is replaced by a new child individual. Note that a child individual, whose structure is identical to any of the individual 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 individual, i.e., the best schedule.

5. Simulation Study

A discrete event simulation was built for testing and evaluation. We used the toolkit [16], [17] to simulate machines, schedulers, network, and collect the statistical data. Ten computational nodes with different processing abilities were also simulated. The communication costs of the tasks follow the Uniform distribution with the mean 20 s. The task sizes follow a real workload trace from [18]. The arrival rate of tasks follow the Logarithmic distribution. Figure 9 is the distribution of 10000 task sizes and Fig. 10 is the number of new task arrivals every 100 s.

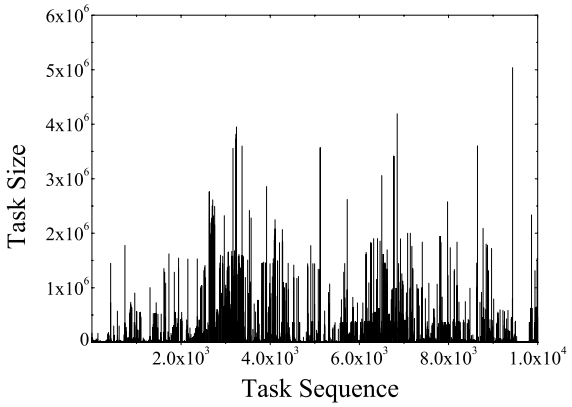

Fig. 9 Task sizes distribution.

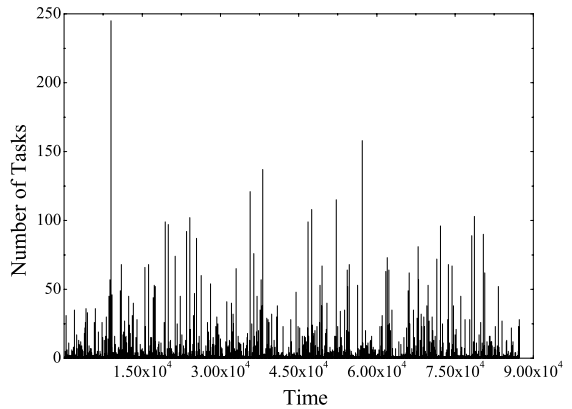

Fig. 10 Task arrivals distribution.

In all experiments the population size is 80. 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 [19]. The execution time t_s of creating a schedule by our genetic algorithm is estimated by the following conservative estimate equation.

$$t_s = 40 \exp \frac{|N|}{1580} - 40. \quad (4)$$

The error of this estimate equation is less than 10% in the worst case.

5.1 Scheduling Algorithm Comparison

Our genetic algorithm is designed to shorten the completion time of a batch of tasks. We choose the MaxMin, MinMin, Sufferage, and GA [6] to compare with our genetic algorithm.

In this experiment all algorithms deal with the first 5000 tasks in Fig. 9, and all 5000 tasks are in only one batch for all algorithms. The results are shown in Fig. 11.

In fact there are many task scheduling algorithms for heterogeneous distributed computing. According to the results in [4], MaxMin and MinMin are sensitive to the task size distribution, and Sufferage is the compromise between

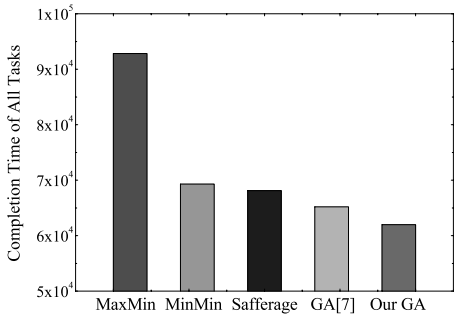

Fig. 11 Comparison of scheduling algorithms.

Table 2 Schedulers.

Scheduler	Strategy	Algorithm
Ours	Dynamic scheduling cycle	Our GA
MM_Time	Regular time interval	MinMin
MM_Count	Fixed Count	MinMin
MCT	FCFS	MCT

MaxMin and MinMin. GA can always achieve the best scheduling result, but its execution time is very long especially for a large batch of tasks [4]–[6]. In the comparisons [6] MinMin was concluded to be the most efficient one in view of the tradeoff between the goodness of scheduling result and the execution time. We only show one figure of the comparison results which demonstrates the advantage of our genetic algorithm in this static environment, because the keystone of this paper is to highlight the advantage of the whole scheduler with our scheduling strategy and genetic algorithm.

5.2 Schedulers Comparison

We compared our strategy and algorithm with the scheduling algorithm, MinMin, and the most familiar scheduling strategies, the regular time interval strategy and the fixed count strategy. Since the immediate mode task scheduling algorithm itself is a scheduling strategy, which has the fastest response to task arrivals, we also choose MCT algorithm, a good immediate mode algorithm [4]–[6] to compare with. We built four schedulers with all strategies and algorithms. All schedulers are listed in Table 2.

The 10000 tasks and the variant arrival rate of tasks in Figs. 9 and 10, form a complex scenario, where the regular time interval is 20 s and the fixed count is 40. In this complex scenario we focus on the waiting time, the execution time, and the response time of tasks and their averages. The waiting time is from the task arriving at the scheduler to the finish time of this task, which mainly consists of the waiting time in the scheduler queue and the waiting time in the local queues. The response time is the sum of waiting time and execution time. Figure 12 is the total waiting time of each task and the waiting time in local task queues. Because the scheduler MCT uses the FCFS strategy, the waiting time in local task queues of MCT is always approximatively equal to the total waiting time and the waiting time in the sched-

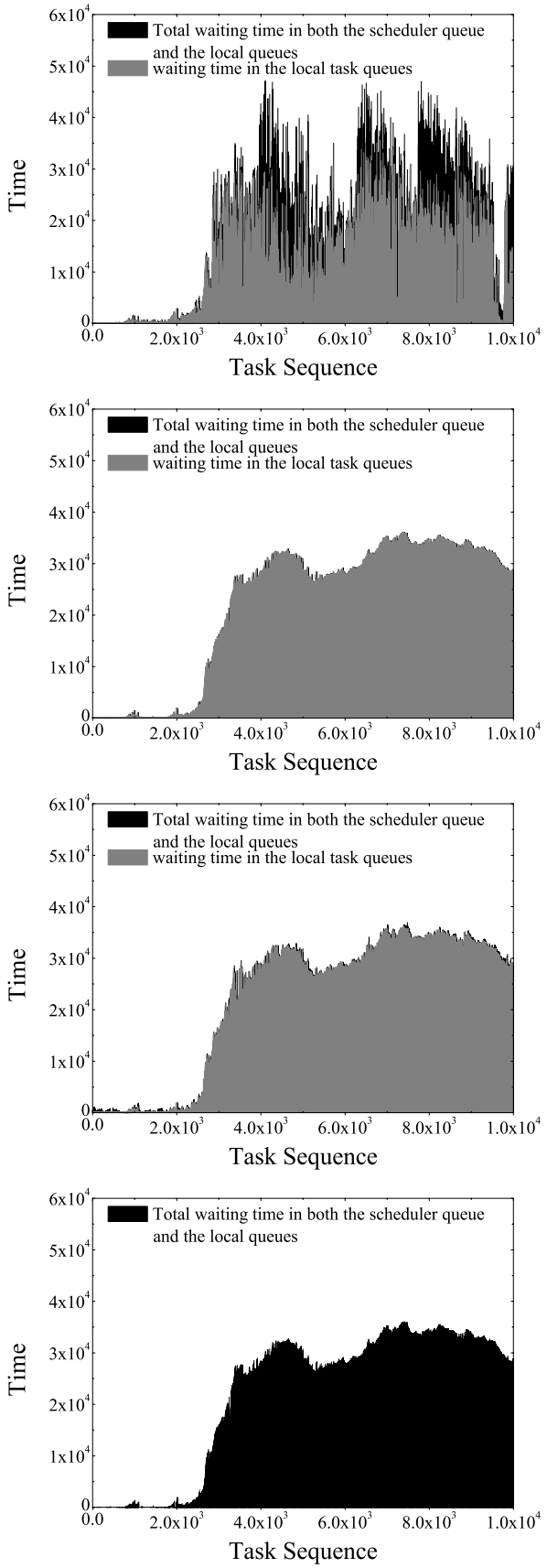

Fig. 12 Waiting time of each task. From the top down the schedulers are Ours, MM_time, MM_Count, and MCT.

uler task queue is always 0. Hence the waiting time in local task queues of MCT is not shown in Fig. 12. Figure 13 is the response time and execution time of each task. The average data are shown in Fig. 14, where it is clear shown that our scheduler achieves the best average scheduling results for the dynamic task flow. The average waiting time, the average execution time, and the average response time of our scheduler are less than those of the others.

5.3 Discussion

The advantage of our scheduler mainly comes from two aspects as follows:

- When many tasks have been waiting in the local task queue, the new arriving tasks will wait in the scheduler task queue so that our strategy can collect more tasks for our genetic algorithm. Because our genetic algorithm can create the schedule with shorter completion time, so the cumulative effect of many schedules leads to the whole superiority. Our genetic algorithm is performed in parallel with the execution of tasks on computational nodes, so the long execution time of genetic algorithm is not a drawback for our scheduler.
- Our scheduler have more opportunities to perform combinatorial optimization on the tasks waiting in the scheduler task queue than the tasks waiting in the local task queues. Our strategy always calls scheduling algorithm to allocate tasks on the best occasions, which means that our scheduler can accumulate much larger batch of tasks without any waste of processing capacity.

The number of tasks in each schedule are shown in Figs. 15 and 16 for our scheduler and MM_Time. The number of tasks in each schedule for the fixed count strategy is always 40, and for FCFS strategy it is always 1. Obviously the numbers of tasks in the schedules created by our scheduler are much larger than those created by the other schedulers, and the frequency of creating schedules is much smaller. It is due to the large batch size that our genetic algorithm can perform the scheduling optimization, which means that the completion time of those large batches of tasks is short. Thus the subsequent tasks will wait shorter time. Hence, the average waiting time of all tasks in local task queues for our scheduler is quite small, and the average waiting time in scheduler queue is large in Fig. 14. But the total waiting time is saved so that the average total waiting time of our scheduler is much smaller.

It is different from our scheduler in Fig. 12 that the waiting time in local task queues of each task for the other three schedulers is always very close to the total waiting time, so for MM_Time and MM_Count the total waiting time in black is almost covered by the waiting time in local task queues in grey. The particular behaviors of all schedulers are depicted in Figs. 12 and 13. Because of the bad adaptability to the dynamic task flow, the other three schedulers' behaviors look same with each other.

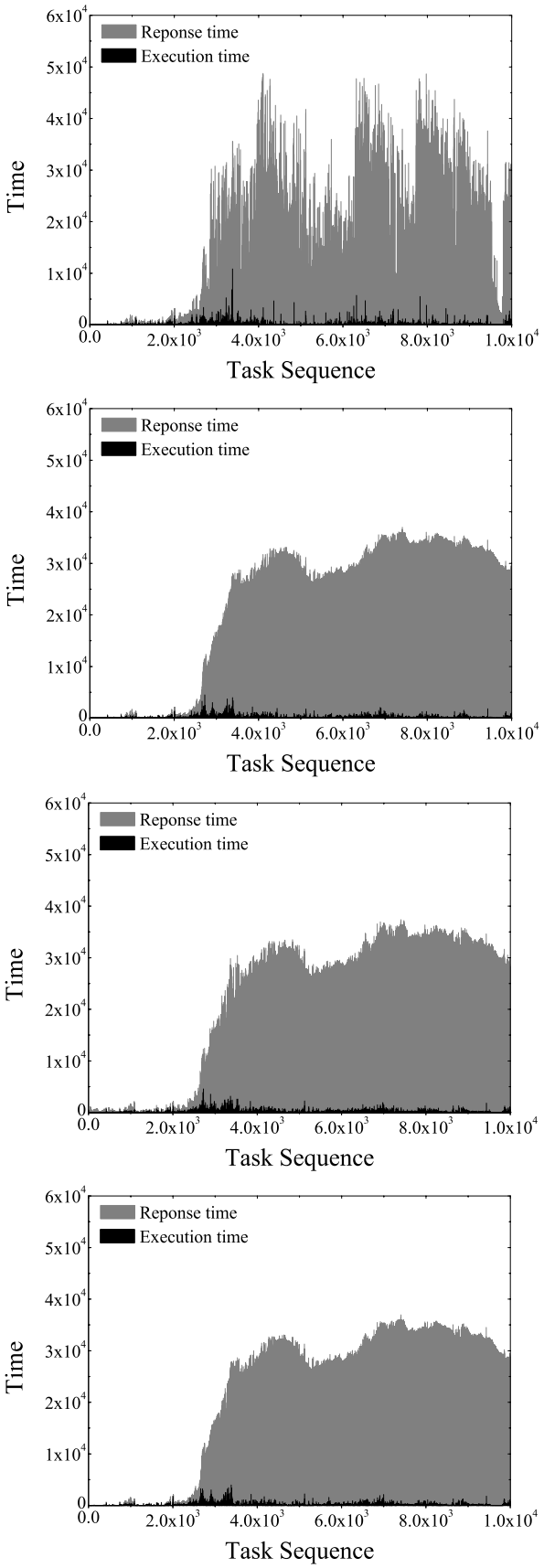

Fig. 13 Response time and execution time of each task. From the top down the schedulers are Ours, MM_time, MM_Count, and MCT.

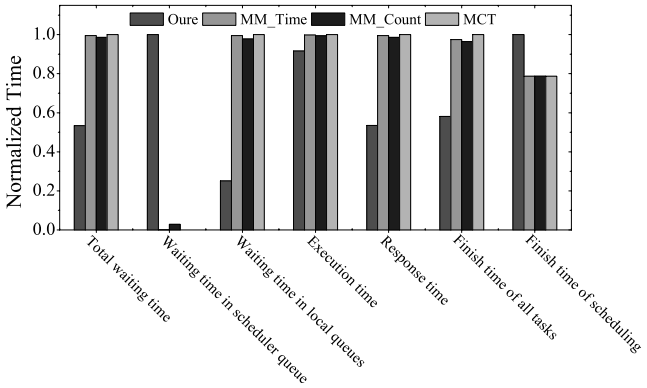

Fig. 14 Normalized average total waiting time, average waiting time in scheduler queue, average waiting time in local queues, execution time, average response time, finish time of all tasks, and finish time of scheduling process for each scheduler. Each group of data is normalized independently.

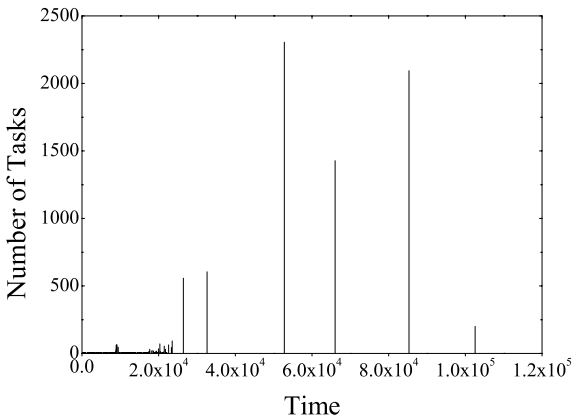

Fig. 15 The number of tasks in each schedule created by our scheduler.

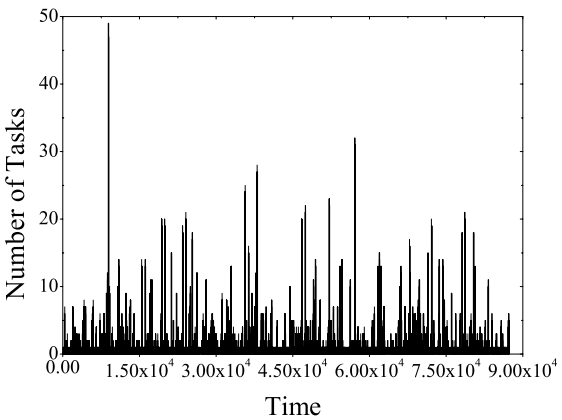

Fig. 16 The number of tasks in each schedule created by MM_Time.

The combinatorial optimization on task scheduling can shorten the completion time of tasks as a whole, but can not promise that the execution time, response time, and waiting time of each task can be optimized. For our scheduler, the waiting time, the execution time, and the response time of serval tasks are long. However, the advantages of our sched-

uler, the short average waiting time, the short average response time, the short average execution time, and the short finish time of all tasks, are at the expense of these several tasks. Despite the sacrifice of a few of tasks, the average execution time of all tasks for our scheduler is still a little shorter.

Note that the simulated environments including the task sizes distribution and the task arrivals can not absolutely cover all the realities despite our complex scenario is so close to the real environments. The average arrival rate of tasks in our simulation is so high that the waiting time of the tasks in the tail of the task flow is very long. But the current results have proved and shown the advantage of our scheduler very well for a busy and crowded system. Moreover the time interval of the regular time interval strategy and the number of tasks for the fixed count strategy can be other values. However the schedulers with longer time interval or larger count are obtuse and the schedulers with much shorter time interval and much smaller count will behave like the immediate mode scheduler which is shown to perform worse than our scheduler.

6. Related Work

Task scheduling in distributed and heterogeneous computing environments has been a hot topic for many years. Some excellent scheduling systems have been developed, for example Condor [20] and Legion [21]. These systems are complicated to satisfy diverse requirements of applications. Most of them have a common key part: decision making, which schedules tasks according to the known information. A decision making is a specific scheduling algorithm. Our work is to develop a task scheduling method with small average response time of tasks for local computing resources of heterogeneous distributed computing system under the given precondition.

Usually different scheduling algorithms are used in a scheduling system for different tasks. An important class of scheduling algorithm is the independent task scheduling algorithm, which is usually used to schedule independent tasks to a local set of machines. The independent task mapping techniques have been well summarized and compared in [4]–[6], where the immediate mode scheduling and the batch mode scheduling were discussed. 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 [4].

It is an inevitable trend to induct the arrival of tasks into the research on task scheduling for heterogeneous distributed computing, for example [22], [23], where the arrival rate of tasks is a constant and the computational sizes of tasks are not known in advance. In [10], a strategy named dynamic batch size is used to estimate the $(p + 1)$th batch size after the pth batch has been scheduled. The estimate is only based on the running time of creating a schedule, so this strategy can only achieve a highly efficient scheduler and can not adapt to the dynamic task flow. In this paper our research focused on the variant arrival rate and the computational sizes can be known in advance. Moreover, instead of directly monitoring the arrival rate of tasks, the number of tasks in the scheduler task queue is used to influence the scheduling strategy. The continuous task arrivals are named task stream in [22], which is named task flow in this paper.

It is shown that genetic algorithm is an effective method for task scheduling and can always achieve the shortest completion time of all tasks [4]–[6]. GA was successfully used for task scheduling in [8]–[10].

7. Conclusion and Future Work

In this paper we propose a task scheduling method for heterogeneous distributed computing, which includes a scheduling strategy named dynamic scheduling cycle and a genetic algorithm. The scheduling strategy dynamically calls our genetic algorithm to create task schedules in terms of the task arrivals measured by the number of tasks in the scheduler task queue. The genetic algorithm can achieve the shorter completion time of a batch of tasks. The strategy and the genetic algorithm work with each other to enhance the scheduler's efficiency and performance. According to the results of simulation our scheduler works well in the simulated environment. In each scheduling cycle our scheduler can generally achieve advantage over the other schedulers, and the long-term advantage is obvious.

The average waiting time and average response time of all tasks are much shorter than those of the other schedulers, but these advantages are at the expense of several tasks. In this paper, the advantages of our scheduler are achieved under the precondition that the tasks do not have deadlines, which is assumed there are no time requirements for tasks just like many other research. In the future we plan to study the improvement on the current results considering the tasks with deadlines.

References

[1] R.F. Freund and H.J. Siegel, "Heterogeneous processing," IEEE Comput., vol.26, no.6, pp.184–199, June 1993.
[2] I. Foster and C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers, San Fransisco, CA, 1999.
[3] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," IEEE Trans. Softw. Eng., vol.11, pp.1427–1436, Nov. 1989.
[4] T.D. Braun, S. Ali, H.J. Seigel, D. Hensgen, and R.F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing system," J. Parallel Distrib. Comput., vol.59, pp.107–131, 1999.
[5] T.D. Braun, H.J. Seigel, N. Beck, L.L. Boloni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, D. Hensgen, and R.F. Freund, "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.
[6] T.D. Braun, H.J. Seigel, N. Beck, L.L. Boloni, M. Maheswaran,

A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, D. Hensgen, and R.F. Freund, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," J. Parallel Distrib. Comput., vol.61, pp.810–837, 2001.
[7] W. Sun, Y. Zhang, and Y. Inoguchi, "Practical task flow scheduling for high throughput computational Grid," Proc. 35th Int'l Conf. Parallel Processing Workshop, pp.291–297, 2006.
[8] A.Y. Zomaya and Y.H. Teh, "Observations on using genetic algorithms for dynamic load-balancing," IEEE Trans. Parallel Distrib. Syst., vol.12, no.9, pp.899–911, Sept. 2001.
[9] A.Y. Zomaya, C. Ward, and B. Macey, "Genetic scheduling for parallel processor system: Comparative studies and performance issues," IEEE Trans. Parallel Distrib. Syst., vol.10, no.8, pp.795–812, Aug. 1999.
[10] A.J. Page and T.J. Naughton, "Dynamic task scheduling using genetic algorithms for heterogeneous distributed computing," Proc. 19th IEEE/ACM Intl. Parallel and Distributed Processing Symposium, p.189a, Denver, Colorado, 2005.
[11] A. Chien, B. Calder, S. Elbert, and K. Bhatia, "Entropia: Architecture and performance of an enterprise desktop grid system," J. Parallel Distrib. Comput., vol.63, pp.597–610, 2001.
[12] C. Weng and X. Lu, "Heuristic scheduling for bag-of-tasks applications in combination with QoS in the computational grid," J. Future Generation Computer System, vol.21, pp.271–280, 2003.
[13] J.H. Holland, Adaptation in Natural and Artificial System, Univ. of Michigan Press, 1975.
[14] P.C. Chu and J.E. Beasley, "A genetic algorithm for the generalised assignment problem," Computer Oper. Res., vol.24, pp.17–23, 1997.
[15] G. Syswerda, "Uniform crossover in genetic algorithms," Proc. 3rd Int'l Conf. on Genetic Algorithms, pp.2–9, 1989.
[16] R. Buyya and M. Murshed, "GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing," J. Concurrency and Computation: Practice and Experience, pp.1–32, 2002.
[17] F. Howell and R. McNab, "SimJava: A discrete event simulation package for Java with applications in computer systems modelling," First Int'l Conf. Web-baesed Modelling and Simulation, San Diego, CA, Jan. 1998.
[18] http://www.supercluster.org/research/traces/
[19] B. Rylander, Computational Complexity and the Genetic Algorithm, Ph.D. Dissertation, University of Idaho, USA, June 2001.
[20] D. Wright, "Cheap cycle from the desktop to the dedicated cluster: Combining opportunistic and dedicated scheduling with Condor," Proc. Int'l Conf. on Linux Cluster: the HPC Revolution, June 2001.
[21] S.J. Chapin, D. Katramatos, J. Karpovich, and A.S. Grimshaw, "The legion resource management system," Proc. 5th Workshop on Job Scheduling Strategies for Parallel Processing, vol.1659, pp.162–178, April 1999.
[22] L. He, S.A. Jarvis, D.P. Spooner, H. Jiang, D.N. Dillenberger, and G.R. Nudd, "Allocating non-real-time and soft real time jobs in multiclusters," IEEE Trans. Parallel Distrib. Syst., vol.17, no.2, pp.99–112, Feb. 2006.
[23] V. Berten, J. Goossens, and E. Jeannot, "On the distribution of sequential jobs in random broking for heterogeneous computational Grids," IEEE Trans. Parallel Distrib. Syst., vol.17, no.2, pp.113–124, Feb. 2006.

Wei Sun received his B.E. and M.E. degrees from Tianjin University, China, in 1998 and 2005. From 1998 to 2002 he was an engineer in the Sixth Research Institute (Electronics) of Ministry of Information Industry of China. He is currently a PhD candidate at JAIST. His research deals with large scale distributed system, parallel and heterogeneous computing.

Yuanyuan Zhang received the B.E. degree in School of Mechano-Electronic Engineering, and M.E. degree in School of Computer Science and Technology from Xidian University, China, in 2000 and 2003, respectively. She received Ph.D. from Graduate School of Information Science, JAIST (Japan Advanced Institute of Science and Technology) in 2006. She is a researcher of Fujitsu Laboratory Ltd since 2006. Her current research interest is about resource management and information service in grid computing.

Yasushi Inoguchi received his B.E. degree from Department of Mechanical Engineering, Tohoku University in 1991, and received M.S. degree and Ph.D from Japan Advanced Institute of Science and Technology (JAIST) in 1994 and 1997, respectively. He is currently an Associate Professor of Center for Information Science at JAIST. He was a research fellow of the Japan Society for the Promotion of Science from 1994 to 1997. He is also a researcher of PRESTO program of Japan Science and Technology Agency since 2002. His research interest has been mainly concerned with parallel computer architecture, interconnection networks, and high performance computing on parallel machines. Dr. Inoguchi is a members of IEEE and IPS of Japan.