

## Real-Time Gang Schedulings With Workload Models for Parallel Computers\*

WEI-KUAN SHIH, CHUNG-DER LIN, YAR-WEN CHANG

AND JENQ-KUEN LEE

*Department of Computer Science*

*National Tsing-Hua University*

*Hsinchu, Taiwan 300, R.O.C.*

Gang scheduling has recently been shown to be an effective task scheduling policy for parallel computers because it combines elements of space sharing and time sharing [10, 20]. In this paper, we propose new policies to enable gang scheduling to adapt to environments with real-time constraints. Our work, to our best knowledge, is the first attempt to address these real-time aspects with gang scheduling. Our system, guided by a metric called the "task utilization workload," can schedule both real-time and non-real-time tasks at the same time. In this paper, we report simulation results obtained using a family of scheduling algorithms based on our proposed metric. Our scheme is designed for practical use with large scale industrial and commercial parallel systems. Preliminary simulation results also show that our proposed policy is effective for real-time scheduling and can schedule non-real-time tasks with fairness and good throughput.

**Keywords:** real-time scheduling, gang-scheduling, parallel softwares, hierarchical and distributed control, multiprocessor scheduling algorithm

### 1. INTRODUCTION

With advances in parallel computer architectures, parallel machines can now provide the aggregate computing power necessary for large-scale scientific and industry applications. Meanwhile, the lack of system software support for parallel environments has prompted many research and industry efforts in an attempt to improve the ease of use of parallel environments. The most notable recent developments and research efforts in parallel system software include works focusing on parallel languages and environments, such as MPI [6], PVM [4], HPF [13], pC++ [1, 17] etc., on optimizing compilers and library support [12, 16, 23, 24] to optimize program performance, and on task scheduling [10, 20] for parallel operating systems or runtime libraries. As a result of the large number of academic research efforts along with industrial development efforts, many parallel systems are now gradually finding their way into use in industrial and commercial applications in addition to conventional large scale scientific applications. For example, parallel machines with balanced I/O performance [15] can now be used with high-end Web servers and possibly in data-mining processing [7, 8].

---

Received April 12, 1999; revised August 4, 1999; accepted October 19, 1999.

Communicated by Pen-Chung Yew.

\*This work was supported in part by National Science Council of Taiwan under grant Nos. NSC88-2215-E-007-034, NSC87-2215-E-007-014, NSC87-2213-E-007-027 and NSC86-2213-E-007-043.

While the recent advances in system software supporting message-passing libraries, parallel languages, optimizing compilers, and task schedulings have helped move parallel computing into new commercial and industrial applications, we argue that additional key supports are needed to solidify the place of parallel computers in this newly expanded territory. In our opinion, one of the key features needed is real-time support, as most of the industrial systems needing for high-end performance often require real-time support. Examples include parallel computers used for radar and DSP applications, and servers for wireless communications. It, however, remains an open question how real-time support can actually be incorporated into modern parallel system software which include languages, compilers, and schedulers. In this paper, we address this aspect of real-time support for a class of important task scheduling policies, known as gang scheduling, for parallel environments.

In a parallel system, there is a set of parallel tasks. Tasks need to communicate with each other. To resolve the issue of synchronization of tasks, especially in fine-grain interactions, it is best if the scheduler can guarantee that tasks can be executed in the processors simultaneously. Gang scheduling is an efficient way of scheduling interacting tasks or threads because it can guarantee that a number of interacting tasks can run on distinct processors simultaneously [10]. In addition, recently, a mechanism called distributed hierarchical control (DHC) [9] has often been employed to perform gang scheduling. On the other hand, a real-time system will have tasks with additional timing constraints, which need to be satisfied. In our model, there is a system of preemptable, parallel tasks  $T = \{T_1, T_2, \dots, T_n\}$ , in which gang size and, possibly, timing constraints are specified. There are two kinds of tasks in the system, real-time or non-real-time, depending on whether or not deadlines exist. Real-time tasks need to finish execution before deadlines; otherwise, the results will be useless or some penalties may apply. The response times of real-time tasks are not important. Non-real-time tasks don't have deadlines, minimizing the response time and improving the fairness of resource usage are the prime goal when we schedule non-real-time tasks. In addition, reducing waste in idle processors due to gang scheduling is also crucial for the performances of our system.

The key strategies presented in this paper are outlined as follows. First, our system is priority driven, and tasks are on-line tasks. We use a metric called "the task utilization workload" to guide the scheduling of both real-time and non-real-time tasks. In our proposed approach, the current demands of real-time tasks are used as a metric to estimate how many slack time is available for executing non-real-time tasks. The slack time is the maximum amount time we can use to schedule non-real-time tasks before the real-time tasks miss their deadlines. If the amount of slack time is small, we need to execute the real-time tasks immediately. Second, to schedule real-time and non-real-time tasks properly, our system provides three queues. The first one is a real-time queue, used by real-time tasks. The second one is a non-real-time queue, in which non-real-time tasks exist. The third queue is used to prevent starvation of tasks that need large processor gangs. When a non-real-time task request large number of processors and idle for a long time, it is moved into the queue with higher priority to avoid starvation. Finally, a real-time task is scheduled using the early deadline first (EDF) scheme, and a non-real-time task is scheduled based on fairness or priority as set earlier by the system when the threshold of our proposed workloads indicate that a non-real-time task should be scheduled. Non-real-time tasks are given quantum to be scheduled, and when they are context switched, remapping scheme is allowed to reduce the waste in gang-scheduling [20].

We obtained simulation results for a family of scheduling policies using our proposed metric called the “task utilization workload” and determined their characteristics. Our simulations incorporated both distributed hierarchical control (DHC) and arbitrary allocations (where processor allocations are allowed whenever there are idle processors). Preliminary simulation results show that our policy is an effective scheme for performing real-time scheduling, while it can schedule non-real-time tasks with fairness and good throughput. Currently, we are using our scheduling framework on a 32 node Mercury Race computers and 80-node IBM SP-2 accessible to us for performing actual parallel real-time computations, such as for radar and DSP applications, and for server use for wireless communications.

The remainder of the paper is organized as follows. Section 2 gives a precise definition of the system configuration for our model. Next, section 3 presents our scheduling policies for gang scheduling with real-time tasks. Section 4 presents the experimental results. Finally section 5 discusses the related work, and section 6 concludes this paper.

## 2. SYSTEM CONFIGURATION

Given a system of preemptable, parallel tasks  $T = \{T_1, T_2, \dots, T_n\}$  in which each task  $T_i$  is characterized by the following parameters, which are rational numbers:

- $d_i$ : deadline of  $T_i$  by which time  $T_i$  must be completed. This is for real-time tasks only.
- $e_i$ : the execution time of task  $T_i$ , which is the time required to execute  $T_i$  to completion in the traditional sense.
- $g_i$ : the gang size, the number of processors needed for execution by task  $T_i$ .

There are two kinds of tasks in the system. Tasks are real-time or non-real-time, depending on whether or not deadlines exist. Real-time tasks need to finish execution before deadlines; otherwise, the results will be useless, or some penalties may apply. The response times of real-time tasks are not important. Non-real-time tasks don't have deadlines, so minimizing the response time and improving the fairness of resource usage are the prime goal when we schedule non-real-time tasks. The number of available processors in the system, denoted as  $m$ , is usually a power of 2, such as 128 or 256.

The system is priority driven; that is, processors never idle when there are tasks ready to and can be executed. To schedule real-time and non-real-time tasks properly, our system provides three queues. The first one is the real-time queue, used by real-time tasks. The second one is a non-real-time queue, in which the non-real-time tasks exist. The third one is a semi-real-time queue. The job of the third queue is to prevent starvation of tasks that need large processor gangs. When a non-real-time task requests a large number of processors, there may not be enough processors available for quite a long time. It is possible that this task can never be executed. To avoid starvation of this task, we temporarily move it into semi-real-time queue when it has been waiting for a long time, and we schedule it as a real-time task. While the task is in the semi-real-time queue, the deadline is set to the earliest deadline of all the real-time tasks. After the task is executed once, it is moved back to the non-real-time queue.

The scheduling policy that the schedule will apply whenever it needs to make a scheduling decision is presented in the next section.

### 3. SCHEDULING POLICY

In our system, there are three queues. We need a good scheduling policy to help the scheduler make decisions whenever there are processors available for executing tasks. The goal of our scheduler is to find a good schedule in which real-time tasks are completed before their deadlines and non-real-time tasks are completed as soon as possible. Moreover, we also want to achieve the goal of gang scheduling so that wastes will be minimized and fairness of resource usage will be guaranteed. To achieve this goal, we can not schedule real-time tasks too early or too late. If the scheduling policy is to schedule real-time tasks as early as possible, the number of real-time tasks that miss their deadlines is minimized, but the non-real-time tasks will have bad response time. On the other hand, if we try to improve the response time of non-real-time tasks by delaying the execution of real-time tasks, the number of real-time tasks that miss their deadlines might increase. This is a trade-off between the execution of real-time and non-real-time tasks. In this paper, we propose several approaches to finding the best scheduling policy for scheduling such a system.

In our scheduling policy, one way to resolve this problem is to find the maximum amount time we can use to schedule non-real-time tasks before the real-time tasks miss their deadlines. This amount of time is called the slack time of the real-time tasks. Due to the complicated nature of gang-scheduling, it is impossible to compute the exact amount of slack time for each real-time task. To estimate current amount of slack time in our multi-processor system, we use the current demands of real-time tasks as a metric to estimate how many slack time is available for executing the non-real-time tasks. If the amount of slack time is small, we need to execute the real-time tasks immediately. We set a threshold for the decision making process. If the current demand of real-time tasks is less than the threshold, it is safe to schedule non-real-time tasks for execution. If the current demand of real-time tasks is higher than the threshold, we must execute the real-time tasks immediately; otherwise, some real-time tasks might miss their deadlines. In this approach, we do not compute the demand of real-time tasks directly. Instead, we compute the total utilization of real-time tasks. To measure the current demand of real-time tasks, we propose two different formulas to calculate the total utilization of real-time tasks. They are even distribution utilization and uneven distribution utilization formulas, respectively. We will give the details idea in the next subsection.

#### 3.1 Even Distribution Utilization (EDU)

The most simple way to compute the total utilization of real-time tasks is to compute the utilization of individual real-time tasks first and to then add them all together. In this way, we can evenly distribute the execution time of tasks based on a feasible scheduling interval. It is obvious that we favor real-time tasks for execution than non-real-time tasks in this approach. If no new on-line tasks arrive later, the total utilization decreases whenever a real-time task finishes. When an on-line task arrives, this formula is updated immediately to reflect the current demand of real-time tasks, therefore, the scheduler has a better chance to

finish each on-line real-time task before its deadline. First, we will give a formal definition for Even Distribution Utilization for a single task. After that, we will define Even Distribution Utilization for a set of real-time tasks.

**Definition 1:** Even Distribution Utilization of task  $T_i$ , denoted by  $u_i$ , is defined as follows:

$$u_i = (e_i * g_i) / ((d_i - current\_time) * m),$$

where  $g_i$ ,  $e_i$ , and  $d_i$  are the same as in section 2 for system configurations, and  $m$  is the number of processor in the system. In this definition, we assume that the demand of task  $T_i$  is evenly distributed to the feasible scheduling interval of  $T_i$  (i.e., the feasible scheduling interval of  $T_i$  is the interval (current time,  $d_i$ ) to which  $T_i$  can be scheduled). After  $u_i$  is calculated, we define the Even Distribution Utilization of a set of real-time tasks as follows:

**Definition 2:** Even Distribution Utilization of task set  $T$ , denoted by  $U$ , is defined as follows:

$$U = \sum_{1 \leq i \leq n} (u_i).$$

We will use the following example to show how to compute the EDU of a set of real-time tasks. In this example, we have 8 processors. Assume that current time is 0, and that there are three real-time tasks ready for execution. The parameters and timing constraints of these three tasks are listed in Table 1.

**Table 1. EDU of three real-time tasks.**

Task	$d_i$	$g_i$	$e_i$	$(e_i * g_i)$	$u_i$
$T_1$	2	4	2	8	8/16
$T_2$	3	4	3	12	12/24
$T_3$	6	2	4	8	8/48

$$EDU = (8/16 + 12/24 + 8/48) = 7/6$$

In this example, there are three real-time tasks. The first task, denoted by  $T_1$ , has deadline 2, execution time 2 and gang size 4. The second task  $T_2$  has deadline 3, execution time 3 and gang size 4. The last task  $T_3$  has deadline 6, execution time 4 and gang size 2. Assume that the current time is 0. According to our formula, the utilization of  $T_1$ ,  $T_2$  and  $T_3$  is 8/16, 12/24 and 8/48, respectively. Therefore, the EDU of this task set at time 0 is  $U = (8/16 + 12/24 + 8/48) = 7/6$ . Since we evenly distribute the real-time demand of each real-time task into its feasible scheduling interval first and then add them together, it is possible that the EDU will be larger than one. This doesn't mean that the system is overloaded. In our example, when the current time is 0, the EDU is large than one. Therefore, we execute real-time tasks immediately. The EDU is just a metric used to measure how urgent real-time tasks need to be executed.

### 3.2 Uneven Distribution Utilization (UDU)

In the second approach, we try to measure the demand of the real-time tasks by distributing the execution time of real-time tasks to their feasible scheduling intervals in an uneven way. We mean “uneven” from the point of view of single real-task. If we treat all the real-time tasks together, we actually try to distribute their real-time demands into their feasible time intervals as even as possible. The utilization computed in this way can truly reflect the actual system load, but the risk that real-time tasks will miss their deadlines is increased. In other words, when we compute the total utilization of real-time tasks at the current time, the tasks with earlier deadlines will contribute more and that with late deadlines will contribute less. Before we explain how to compute the UDU of all real-time tasks, we will first define the UDU of individual tasks as follows:

**Definition 3:** Uneven Distribution Utilization of task  $T_i$ , denoted by  $u_i$ :

$$u_i = \sum_{j \leq i} (e_j * g_j) / ((d_i - \text{current\_time}) * m),$$

where  $g_j$ ,  $e_j$ , and  $d_i$  are the same as the one used in section 2 for system configurations, and  $m$  is the number of processors in the system. When we compute the UDU of task  $T_i$ , we assume that the deadlines of tasks with deadlines earlier than  $d_i$  can be extended to  $d_i$ ; therefore, the real-time demands of tasks with deadlines earlier than or equal to  $d_i$  can be evenly distributed to the time interval (current time,  $d_i$ ).

**Definition 4:** Uneven Distribution Utilization of task set  $T$ , denoted by  $U$ :

$$U = \text{MAX}_{1 \leq i \leq n} (u_i).$$

The UDU of the set of real-time tasks is the maximum  $u_i$  among all the individual real-tasks. We will use the same example to show how the EDU of a set of real-time tasks is computed. The parameters and timing constraints of these three tasks are the same as before. Their UDU is listed in Table 2.

**Table 2. UDU of three real-time tasks.**

Task	$d_i$	$g_i$	$e_i$	$u_i$
$T_1$	2	4	2	8/16
$T_2$	3	4	3	20/24
$T_3$	6	2	4	28/48

$$\text{UDU} = \text{MAX} (8/16, 20/24, 28/48) = 20/24$$

There are three real-time tasks in this example. Assume that the current time is 0. According to our formula, the UDU of  $T_1$ ,  $T_2$  and  $T_3$  is 8/16, 20/24 and 28/48, respectively. Therefore, the UDU of this task set is  $U = \text{MAX} (8/16, 20/24, 28/48) = 20/24$ .

### 4. EXPERIMENT RESULTS

In this section, we will report simulation results obtained using our proposed scheduling policy. Fig. 1 and Fig. 2 show the results for our first test set. The data set used had normal distribution, and the system was assumed to have 128 processors. The parameters used are listed below:

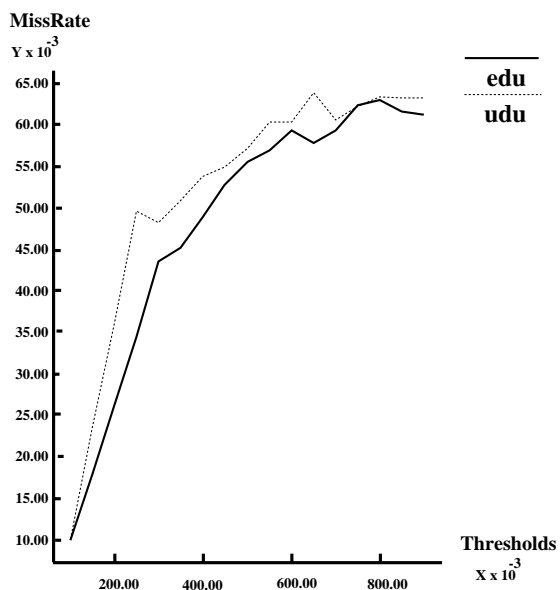


Fig. 1. MissRate: EDU vs. UDU.

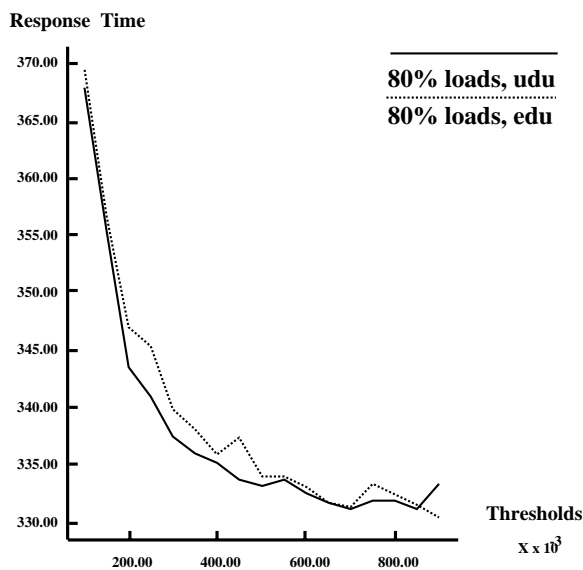


Fig. 2. Response time of non-real-time: EDU vs. UDU.

arrival time: mean = 50, variance = 15;  
 task length: mean = 320, variance = 120;  
 gang size: mean = 16, variance = 8;  
 deadline/task-length:

mean = 1.8, variance = 0.3;  
 50% real-time tasks;  
 50% non-real-time tasks;  
 80% loaded.

Fig. 1 shows the miss rate of real-time tasks when threshold grows. The scheduling policies we used to obtain in the result shown in Fig. 1 employ the EDU and UDU models. Fig. 2 shows the response time of non real-time tasks with increasing thresholds under the EDU and UDU models, respectively. Both of the models show that the response time of non real-time tasks improve as the threshold grows. This threshold is the same one we use to estimate the urgency with which we need to schedule real-time tasks. When we find that it is not urgent to schedule real-time tasks, we can schedule non real-time tasks first to improve their response time. There is a tradeoff between the scheduling of real-time and non real-time tasks. The threshold is the guideline for finding a tradeoff. In our experiment, if the real time task utilization rate was lower than the threshold, we could assume that the probability that real-time tasks would miss their deadlines was still low. In this case, we could perform non real-time tasks to improve the system throughput and the response time of non real-time tasks.

A second data set showing the properties which the algorithms have in common is shown in Fig. 3 and Fig. 4. The second test set also had normal distribution, and the parameters were as follows. The system was assumed to have 128 parallel processors and to be 85% loaded. In addition, the variance of the distributions in this set was smaller than that in the first set:

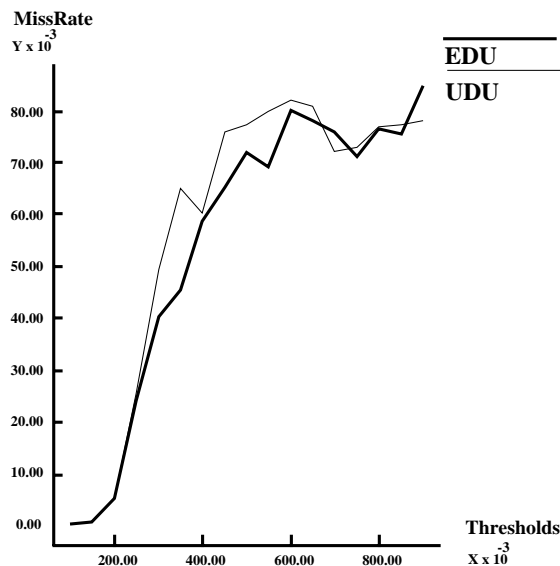


Fig. 3. MissRate: EDU vs. UDU.

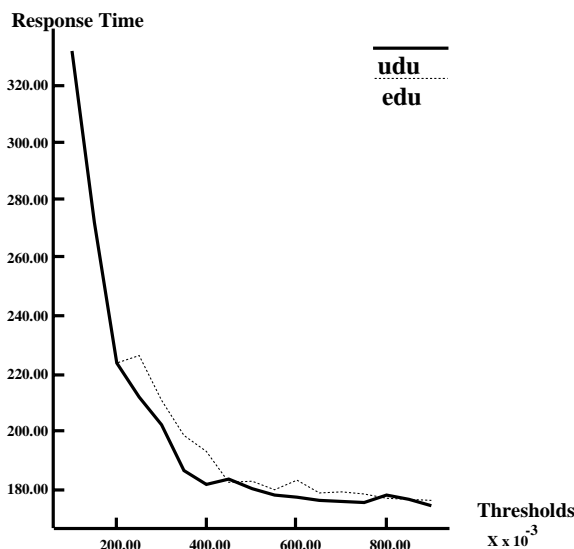


Fig. 4. Response Time: EDU vs. UDU.

arrival time: mean = 50, variance = 5;  
 task length: mean = 173, variance = 12;  
 gang size: mean = 32, variance = 4;  
 deadline/task-length:

mean = 1.8, variance = 0.3;  
 50% real-time tasks;  
 50% non real-time tasks;  
 85% loaded.

Again, Fig. 3 shows the miss rate of real-time tasks when the threshold grows for EDU and UDU models, respectively. Fig. 4 shows the response time of non real-time tasks with increasing thresholds under the EDU and UDU models, respectively. This information represents a first step in characterizing these two models and is, thus, useful. For example, if a real-time system needs to guarantee QoS (quality of service) with a miss rate under 5%, we can utilize the characteristics shown of Fig. 3 to find the best response time of non real-time tasks while satisfying the QoS requirements as well.

Focus is now directed to Figs. 5, 6, 7, and 8, where we show the characteristics of the proposed policy when the workload of the system increases. The workload of a real-time system is an important factor determining the behavior of the real-time system. When the workload of the system increases, the urgency of real-time tasks also increases. In this case, to guarantee that the real-time tasks will meet their deadlines, the scheduler prefers to schedule real-time tasks first. The test set was the same as the first test set, but the workloads of the system were set to 77.5%, 80%, 82.5%, and 85%, respectively. Figs. 5 and 6 show the miss rate of real-time tasks under the EDU and UDU models, respectively. As seen in the figures, the miss rate of real-time tasks increases when the workload of the real-time system

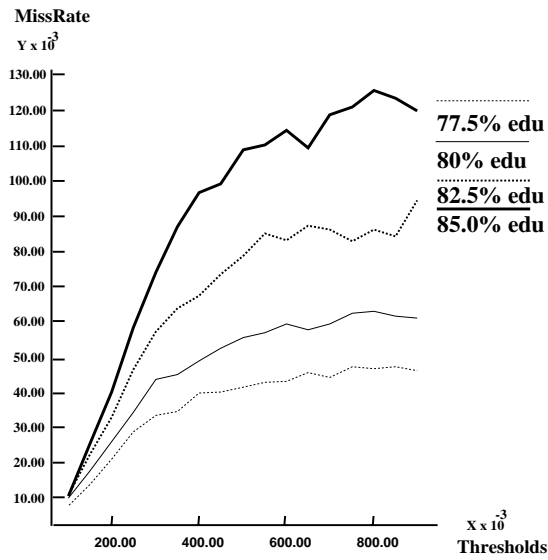


Fig. 5. MissRate of EDU with varied loads.

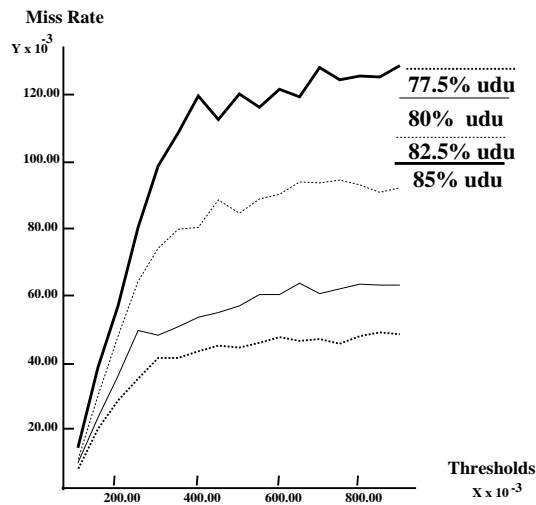


Fig. 6. MissRate of UDU with varied loads.

increases. Figs. 8 and 9 show the response time of the non real-time tasks under the EDU and UDU models, respectively. As seen in this figure, the response time of the non real-time tasks increases when the workload of the real-time system increases. To keep the response time of the non real-time tasks within a reasonable range, we need to increase the thresholds. The drawback of increasing the threshold is that the miss rate of the real-time tasks will increase. All of the above scheduling is based on arbitrary processor allocations; i.e.,

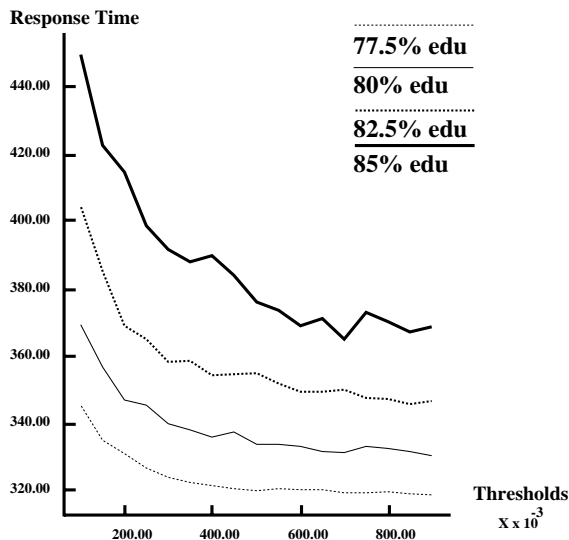


Fig. 7. Non-real-time task response of EDU with varied loads.

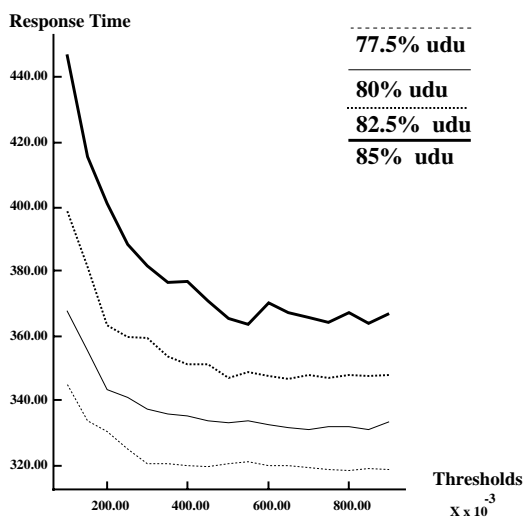


Fig. 8. Non-real-time task response of UDU with varied loads.

processor allocation is allowed whenever there are idle processors. In addition, we show our processor allocation algorithms based on distributed hierarchical control (DHC) [9]. Fig. 9 shows the preliminary performance results when EDU and DHC to work together. The second data test was used, the number of processor nodes was set to 128, and the work load was set to be 60%. In addition, the one with higher miss rate is with 70% real-time tasks in the system, while the other one is with 50% real-time tasks in the system. Both of them exhibit

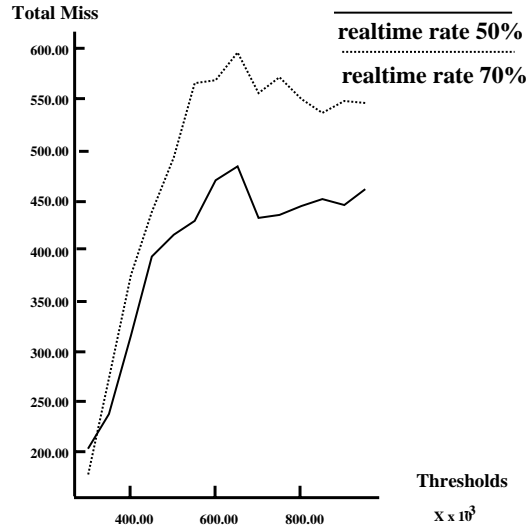


Fig. 9. Miss Total with EDU and distributed hierarchical control.

behavior similar to those in our earlier experiments. Finally, we want to point out that our proposed schemes are superior to a naive scheme which always schedules real-time tasks with highest priority. The response time of non-real-time tasks with this naive scheme will be the slowest one among the data reported in Fig. 4 and etc. Significant improvement can be made by incorporating our proposed workload model.

## 5. RELATED WORK

Our work deals with on-line gang scheduling for real-time tasks. In the related work, Baruah et al. [5] gave the on-line real-time scheduling and gave a nice upper bound on the best performance on-line real-time scheduling can get. Shih et al. [21, 22] gave several on-line algorithms for scheduling real-time tasks in the imprecise computation model. Liu et al. [19] studied the periodic task scheduling and gave an elegant worst case utilization bound for the famous “rate monotonic” scheduling algorithm. All of the important results above with real-time schedulings are for a single processor.

In the work related to scheduling in parallel processors, the work in [11] proposed a scheduling algorithm – the list scheduling algorithm for scheduling tasks in a multi-processor system. They also derive two nice approximation bounds for the worst performance of this algorithm. This work is for parallel and on-line environments, but not for real-time systems. The work in [2] uses the network flow to find an optimal schedule for a multi-processor system to minimize mean weighted execution time of real-time tasks. In addition, the work in [3] is an excellent survey for real-time scheduling on parallel computers. However, none of the above work deals with the issues with gang schedulings. Gang scheduling is a mechanism to resolve the synchronization issue of tasks, especially in the fine-grain interactions. The scheduler can guarantee that tasks can be executed in the processors simultaneously.

The work done in gang scheduling for parallel machines can be seen in [9, 10]. Recently, remapping is used to improve the system utilizations in gang schedulings [20]. These work with gang scheduling did not deal with the issues with real-time systems. Kwon et al. [14] addressed the issues with task scheduling in hypercube systems. Their work only deals with admission controls to decide if the arrival of a real-time task should be granted admission for scheduling. Their system did not handle DHC mechanism, and can't schedule non-real-time tasks.

## 6. CONCLUSIONS

Gang scheduling has recently been shown to be an effective task scheduling policy for parallel computers because it combines elements of space sharing and time sharing. In this paper, we have proposed a new policy which can enable gang scheduling to adapt to environments with real-time constraints. This, to our best knowledge, is the first work to address the real-time aspects of gang scheduling. Our system, guided by a metric called the "task utilization workload," can schedule both real-time and non real-time tasks at the same time. In this paper, we have reported simulation results obtained using a family of scheduling algorithms based on our proposed metric. In our scheduling algorithm, we use thresholds to control the tradeoff between real-time and non real-time tasks. Preliminary simulation results also show that our proposed policy is an effective scheme for performing real-time scheduling while it can schedule non real-time tasks with fairness and good throughput.

Our scheduling algorithms do not consider the case where the workload of the real-time system is changing dynamically. We also have not considered situations where the real-time tasks have hard deadlines, that is, where real-time tasks can not miss their deadlines. We are now working on these two problems.

## REFERENCES

1. F. Bodin, P. Beckman, D. Gannon, S. Narayana, and S. Yang, "Distributed pC++: basic ideas for an object parallel language," *Scientific Programming*, Vol. 2, No. 3, 1993, pp. 7-22.
2. J. Blazewicz and G. Finke, "Minimizing mean weighted execution time on identical and uniform processors," *Information Processing Letters*, Vol. 24, 1987, pp. 259-263.
3. J. Blazewicz, "Selected topics in scheduling theory," *Annals of Discrete Mathematics*, Vol. 31, 1987, pp. 1-60.
4. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM - Parallel Virtual Machine: A User's Guide and Tutorial for Networked Parallel Computing*, MIT Press, Cambridge, MA, 1994.
5. S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Hasha, and F. Wang, "On the competitiveness of on-line real-time task scheduling," in *Proceedings of the 12th Real-Time Systems Symposium*, 1991, pp. 106-115.
6. A User's Guide to MPI [4] Message Passing Interface Forum, "MPI: a message-passing interface standard," *International Journal of Supercomputer Applications*, Vol. 8, Nos. 3/4, 1994, pp. 159-416.

7. M. V. Joshi, G. Karypis, and V. Kumar, "ScalParC: a new scalable and efficient parallel classification algorithm for mining large datasets," in *Proceedings of IPPS/SPDP '98*, 1998, pp. 573-579.
8. S. Goil and A. Choudhary, "High performance data mining using data cubes on parallel computers," in *Proceedings of IPPS/SPDP '98*, 1998, pp. 548-555.
9. D. G. Feitelson and L. Rudolph, "Distributed hierarchical control for parallel processing," *Computer*, Vol. 23, No. 5, 1990, pp. 65-77.
10. D. G. Feitelson and L. Rudolph, "Gang scheduling performance benefits for fine-grain synchronization," *Journal of Parallel and Distributed Computing*, Vol. 16, No. 4, 1992, pp. 306-318.
11. R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Annals of Discrete Mathematics*, Vol. 5, 1979, pp. 287-326.
12. G. H. Hwang, J. K. Lee, and D. C. Ju, "An array operation synthesis scheme to optimize Fortran 90 programs," in *Proceedings of ACM SIGPLAN Conference on Principles and Practice of Parallel Programming*, 1995, pp. 112-122.
13. C. Koelbel, D. Loveman, R. Schreiber, G. Steele, and M. Zosel, *The High Performance Fortran Handbook*, MIT-press, Cambridge, 1994.
14. O-Hoon Kwon, J. Kim, S. J. Hong, and S. Lee, "Real-time task scheduling in hypercube systems," *International Conference on Parallel Processing*, 1997, pp. 166-169.
15. J. K. Lee, I. K. Tsaur, and S. Y. Hwang, "Parallel array object I/O support on distributed environments," *Journal of Parallel and Distributed Computing*, Vol. 40, 1997, pp. 227-241.
16. J. K. Lee, D. Ho, and Y. C. Chuang, "Data distribution analysis and optimization for pointer-based distributed programs," *International Conference on Parallel Processing*, 1997, pp. 56-63. (Also Received the Most Original Paper Award in ICPP '97).
17. J. K. Lee and D. Gannon, "Object-oriented parallel programming: experiments and results," in *Proceedings of Supercomputing '91*, 1991, pp. 273-282.
18. *Tools, Libraries, and Devices*, Mercury Computer Systems, Inc., 1994.
19. C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of ACM*, Vol. 20, No. 1, 1973, pp. 46-61.
20. J. Y-T. Leung, T. W. Tam, C. S. Wong, and G. H. Young, "Minimizing mean flow time with error constraints," in *Proceedings of the 10th Real-Time Systems Symposium*, 1989, pp. 1-11.
21. S. K. Setia, "Trace-driven analysis of migration-based gang scheduling policies for parallel computers," *International Conference on Parallel Processing*, 1997, pp. 489-492.
22. W. K. Shih, Jane W. S. Liu, and J. Y. Chung, "Algorithms for scheduling imprecise computations with timing constraints," *SIAM Journal on Computing*, Vol. 20, No. 3, 1991.
23. W. K. Shih and Jane W. S. Liu, "On-line algorithm for scheduling imprecise computations," *SIAM Journal on Computing*, 1996, pp. 1105-1121.
24. M. Wolfe, *High Performance Compilers for Parallel Computing*, Addison-Wesley Publishing Company, 1994.
25. H. Zima and B. Chapman, *Supercompilers for Parallel and Vector Computers*, Addison-Wesley Publishing Company, May 1990.



**Wei-Kuan Shih** (石維寬) received the B.S. and M.S. degrees in computer science from the National Taiwan University, and the Ph.D. degree in computer science from the University of Illinois, Urbana-Champaign. He is an Associate Professor in the Department of Computer Science at the National Tsing-Hua University, Taiwan. His research interests include real-time systems, VLSI design automation, and wireless communication. From 1986 to 1988, he was with the Institute of Information Science, Academia Sinica, Taiwan.



**Chung-Der Lin** (林崇德) received the B.S. and M.S. degrees in computer science from National Tsing-Hua University, Taiwan, in 1997 and 1999 respectively. He will join Institute for Information Industry (III) of Taiwan around Dec. 1999 as a staff engineer in designing automatic testing systems for hardware devices.



**Yar-Wen Chang** (張雅雯) received the B.S. degree in National Chiao-Tung University in computer science in 1997, and M. S. degrees in computer science from National Tsing-Hua University, Taiwan, in 1999. She is currently with Taiwan Telecom Ltd. as a staff engineer in wireless computings and telecommunications.



**Jenq-Kuen Lee** (李政崑) received the B.S. degree in computer science from National Taiwan University in 1984. He received a Ph.D. in computer science from Indiana University in 1992, where he also received a M.S. (1991) in computer science. He has been an associate professor in the Department of Computer Science at National Tsing-Hua University, Taiwan, since 1992. He was a key member of the team who developed the first version of the pC++ language and SIGMA system while at Indiana University. He was also a recipient of the most original paper award in ICPP'97 with the paper entitled "Data Distribution Analysis and Optimization for Pointer-Based Distributed Programs". His current research interests include optimizing compilers, parallel object-oriented languages and systems, parallel I/O environments, and parallel problem solving environments.