A Linear Time Online Task Assignment Scheme for Multiprocessor Systems

Almut Burchard [*], *Yingfeng Oh* [**], *Jörg Liebeherr* [**], *and Sang H. Son* [**]
[*] School of Mathematics, Georgia Institute of Technology, Atlanta, GA 30332
[**] Computer Science Department, University of Virginia, Charlottesville, VA 22903

Abstract

In this study, a new online task assignment scheme is presented for multiprocessor systems where individual processors execute the rate-monotonic scheduling algorithm. The computational complexity of the task assignment scheme grows linearly with the number of tasks, and its performance is shown to be significantly better than previously existing schemes. The superiority of our scheme is achieved by a new schedulability condition derived for the rate-monotonic scheduling discipline.

1 Introduction

Rate-monotonic (RM) scheduling is becoming a viable scheduling discipline for real-time systems. Through the years, researchers have successfully applied this discipline to tackle a number of practical problems, such as task synchronization, bus scheduling, joint scheduling of periodic and aperiodic tasks, and transient overload [4, 9]. This is done through developing various scheduling algorithms to cope with situations that are not covered by the rate-monotonic algorithm.

While rate-monotonic scheduling is optimal for uniprocessor systems with fixed-priority assignments, it is, unfortunately, not so for multiprocessor systems. In fact, the problem of optimally scheduling a set of periodic tasks on a multiprocessor system using either fixed-priority or dynamic priority assignments is known to be intractable [6]. Hence, any practical solution to the problem of scheduling real-time tasks on multiprocessor systems presents a trade-off between computational complexity and performance. Heuristic algorithms have been shown to deliver near-optimal solutions with limited computational overhead.

In this study, we are concerned with developing an efficient heuristic algorithm for scheduling a set of periodic tasks on a multiprocessor system. The general solution to such a problem involves two algorithms: one to assign tasks to individual processors, and the other to schedule tasks assigned on each individual processor. If the the entire task set is known a priori, the scheduling method is referred to as being *offline*, otherwise it is said to be *online*.

Since real-time systems often operate in dynamic and complex environments, many scheduling decisions must be made online. For example, a change of mission may require the execution of a totally different task set. Or the failure of some processors may render the re-assignment of tasks necessary. In these scenarios, the entire task set to be scheduled may change dynamically, that is, tasks must be added or deleted from the task set. In the following, we will present an online task assignment scheme for multiprocessor systems where each processor executes the RM scheduling algorithm.

Previous work on this problem illustrates the trade-off between computational complexity and performance of heuristic task assignment schemes. The complexity of an algorithm is given by the upper bound of the time required to schedule a set of K tasks. The performance of task assignment schemes is evaluated by providing worst case bounds for N/N_{opt}, where N is the number of processors required to schedule a task set with a given heuristic method, and N_{opt} is the number of processors needed by an optimal assignment. Bounds for the existing schemes are determined by $\lim_{N_{opt}\to\infty} N/N_{opt}$.

In [2], an algorithm is presented with complexity $O(K)$ and $\lim_{N_{opt}\to\infty} N/N_{opt} = 2.28$. The two scheduling algorithm in [8] have a time complexity of $O(K \log K)$, and worst case performance $\lim_{N_{opt}\to\infty} N/N_{opt} = 2.33$ and 2.66. Both studies apply variants of well-known heuristic bin-packing algorithms where the set of processors is regarded as a set of bins [1]. The decision whether a processor is full is determined by a schedulability condition. Also, these assignment schemes are based on the sufficient schedulability condition for uniprocessor systems derived in [7] and its variants, e.g., [3]. Thus, the existing assignment schemes differ mainly in the choice of the bin-packing heuristic.

Our approach for developing a task assignment scheme for multiprocessor systems is different from previous work. Rather than increasing the level of sophistication of the bin-packing heuristic, we focus on developing tighter schedulability condition that allows us to assign more tasks to each processor. If the periods of tasks are close enough, we will show that each processor can be almost fully utilized. This is achieved with a new schedulability condition. The complexity of our assignment scheme is given by $O(K)$ and the worst case performance bounds is $\lim_{N_{opt}\to\infty} N/N_{opt} = 1/(1-\alpha)$, where α is the maximum load factor of any single task.

2 Task Model and Schedulability Condition

We assume that the real-time computer system consists of a multiprocessor system and a set of K real-time tasks. The multiprocessor and the task set are characterized as follows.

A real-time task is denoted by $\tau_i = (C_i, T_i)$ $(i = 1, \ldots, K)$. T_i denotes the shortest time between two requests of task τ_i, and is referred to as the *period* of τ_i. C_i denotes the maximum execution time of task τ_i. Since we assume that the multiprocessor system is homogeneous the execution time is identical on each processor. Each request for a real-time task must complete execution before the next request of the same task. Thus, in the worst case, the execution of τ_i must be completed after T_i time units. The period and the maximum execution time of task τ_i satisfy

$$T_i > 0, \quad 0 \le C_i \le T_i, \qquad\qquad i = 1, \ldots, k$$

We will refer to $U_i = C_i/T_i$ as the *load factor* of the i-th task, and to $U = \sum_{i=1}^{K} U_i$ as the *total load* of the task set. We define α to be the maximal load factor of each task, i.e., $\alpha \ge \max_{1\le i\le K} U_i$. ρ_n denotes the *utilization* of the n-th processor, that is, the sum of the load factors of the tasks assigned to processor n. Tasks are grouped into M classes, and only tasks from the same class can be assigned to the same processor.

Next we derive a sufficient schedulability condition for a processor that schedules tasks with the RM algorithm. The result, presented in Theorem 1, is a simple modification to the schedulability condition for uniprocessor systems by Liu and Layland [7]. Our condition yields a higher utilization

[1] The bin-packing problem is concerned with packing different-sized items into fixed-sized bins using the least number of bins [5].

of the processor if the task periods satisfy certain constraints. On a uniprocessor system, Theorem 1 does not provide a significant improvement for scheduling real-time tasks. For multiprocessor scheduling, however, we can divide a large task set into subsets in such a way that we can make use of the sharpened condition on all but possibly M processors.

The schedulability condition presented in the following theorem takes advantage of a special property of the RM scheduling algorithm. We show that we can increase the processor utilization if all periods in a task set have values that are close to each other. The proof of the theorem can be found in [1].

Theorem 1 *Given a real-time task set $\tau_1, \ldots, \tau_K$. For $i = 1, \ldots, K$, define*

$$S_i := \log_2 T_i - \lfloor \log_2 T_i \rfloor \qquad \text{and} \qquad \beta := \max_{1 \le i \le K} S_i \; - \; \min_{1 \le i \le K} S_i \tag{1}$$

A task set with $\beta < 1 - 1/K$ can be feasibly scheduled by the Rate-Monotonic algorithm if the total load satisfies (2). The condition is tight.

$$U \le (K-1)\left(2^{\beta/(K-1)} - 1\right) + 2^{1-\beta} - 1 \tag{2}$$

Note that the condition given by (2) is tighter than the one given by Liu and Layland [7] under $\beta < 1 - 1/K$.

Corollary 1 *Given a set of real-time tasks $\tau_1, \ldots, \tau_K$, If the total load satisfies $U \le \max\{\ln 2, 1 - \beta \ln 2\}$, then the task set can be scheduled on one processor, where β is as defined above in (1).*

3 An Online Task Assignment Scheme

Our new scheme is based on the schedulability condition of Theorem 1. The parameter used in for the scheme, M, denotes the number of processors to which a new task can be assigned. Recall that tasks are divided into M classes. The class membership of a task τ is determined by the following expression:

$$m = \left\lfloor M(\log_2(T) - \lfloor \log_2(T) \rfloor) \right\rfloor + 1 \tag{3}$$

Each processor is assigned tasks from only one class. Thus, at each processor the value of β as defined in (1) is bounded above by $1 - \ln 2/M$. For each class, the scheme one so-called *current processor*. If a new task from class m is added to the task set, the scheme first attempts to accommodate the task to the current processor for class m. A complete description of the algorithm for assigning a task $\tau = (C, T)$ is given in Algorithm 1.

In Algorithm 1, adding a new task $\tau = (C, T)$ is accomplished in the following manner. First, the class membership of the new task τ (Step 1) is determined. If τ can be added to the current processor of class m without violating the schedulability condition it is assigned to this processor. Otherwise, τ is assigned to an empty processor. If the load factor of τ is sufficiently small (Step 4), the processor to which τ is assigned becomes the current processor of class m (Step 5). If the load factor of τ is large, no other task will be assigned to this processor (Step 7).

Global functions:

curr(m) – Returns the current processor for class m.

newproc() – Returns the index of an empty processor.

Add ($\tau = (C, T)$)

1. $m := \left\lfloor M\left(\log_2(T) - \lfloor \log_2(T) \rfloor\right)\right\rfloor + 1$;
2. **if** ($\rho_{curr(m)} + C/T \leq 1 - \ln 2/M$) **then**
3. $\rho_{curr(m)} := \rho_{curr(m)} + C/T$;
4. **else if** ($\rho_{curr(m)} < C/T$) **then**
5. $curr(m) := newproc(); \rho_{curr(m)} := C/T$;
6. **else**
7. $x := newproc(); \rho_x := C/T$;
8. **endif**

Algorithm 1. Online Task Assignment.

The performance bounds of our scheme are given in Theorem 2 and Corollary 2. Corollary 2 states the asymptotic bound.

Theorem 2 *If a task set is scheduled by Algorithm 1, then the number of processors needed satisfies (4) if* $\alpha \leq (1 - \ln 2/M)/2$, *and satisfies (5) if* $\alpha \geq (1 - \ln 2/M)/2$. *Both bounds are tight.*

$$N < \frac{U}{1 - \ln 2/M - \alpha} + M \tag{4}$$

$$N < \frac{2U}{1 - \ln 2/M} + M \tag{5}$$

Corollary 2 *Let* $\{\tau_i \mid i = 1, 2, \ldots\}$ *be a given infinite task set. Denote by* $U(k)$ *the sum of the load factors of the first* k *tasks. Denote by* $N_M(k)$ *the number of processors used by Algorithm 1, and by* $N_{opt}(k)$ *the number of processors used by an optimal scheme. If* $\lim_{k\to\infty} U(k) = \infty$ *then we have the asymptotic bounds (6) and (7). The bounds are tight.*

$$\lim_{k\to\infty} \frac{U(k)}{N_M(k)} \geq \max\left\{1 - \ln 2/M - \alpha\ ,\ \frac{1 - \ln 2/M}{2}\right\} \tag{6}$$

$$\lim_{k\to\infty} \frac{N_M(k)}{N_{opt}(k)} \leq \min\left\{\frac{1}{1 - \ln 2/M - \alpha}\ ,\ \frac{2}{1 - \ln 2/M}\right\} \tag{7}$$

From the derived bounds we see that the performance of Algorithm 1 is sensitive to the selection of M, the number of task classes. The asymptotic bounds in (6) and (7) improve for large values of M. However, M also determines the number of current processors, i.e., processors which are not fully utilized. Next we present a method for selecting an appropriate value of M.

Assume that the total load of the task set is known. To find the value of M that gives the best worst-case bound for the number of processors used, we fix the value of U in (5). Since the right hand side of (5) is a strictly convex function of M, we can calculate the unique minimum which is denoted by M^*:

$$M^* = \sqrt{2U \ln 2} + \ln 2 \tag{8}$$

This suggests that we should choose $M \sim \sqrt{U}$. Then we obtain

$$U/N \geq (1 - \ln 2/M - \alpha)(1 - M/N) = 1/2 - O(1/\sqrt{U}) \tag{9}$$

and hence

$$N/N_{opt} \leq 2 + O(1/\sqrt{U}) \tag{10}$$

Similarly, if $\alpha < 1/2$, we can minimize the right hand side of (4) over M and obtain that the optimal choice for M should be as close as possible to

$$M^* = \frac{\sqrt{U \ln 2} + \ln 2}{1 - \alpha} \tag{11}$$

If we choose $M \sim \sqrt{U}$, we obtain with (4) the following bound for the average utilization at each processor.

$$U/N \geq (1 - \ln 2/M - \alpha)(1 - M/N) = 1 - \alpha - O(1/\sqrt{U}) \ . \tag{12}$$

and N/N_{opt} is given by

$$N/N_{opt} \leq 1/(1 - \alpha) + O(1/\sqrt{U}) \tag{13}$$

4 Average-Case Performance Evaluation of the New Scheme

While a worst-case analysis assures that the performance bound is satisfied for any task set, it does not provide insight into the average-case behavior of the assignment scheme. To gain insight into the average-case behavior of Algorithm 1, we conduct some simulation experiments.

Our simulations consider large task sets with $100 \leq K \leq 1000$ tasks. In each experiment, we vary the value of parameter M, the number of task classes. The task periods are assumed to be uniformly distributed with values $1 \leq T_i \leq 500$. The execution times of the tasks are also taken from a uniform distribution with range $1 \leq C_i \leq T_i/2$. Thus, α, the maximum load factor of any task, is given by $\alpha = 1/2$. The performance metric in all experiments is the number of processors required to assign a given task set.

We compare our scheme with the online assignment scheme by Davari and Dhall [2], NF-M. Recall that NF-M also has linear computational complexity. The outcome of the simulation experiments is shown in Figure 1. Since an optimal task assignment cannot be calculated for large task sets, we use the total load ($U = \sum_{i=1}^{K} U_i$) to obtain a lower bound for the number of processors required. The maximum number of task classes is set to M = 10, 20, 30, respectively. Each data point in the figure depicts the average value of 15 independently generated task sets with identical parameters. Note that for all values of M, our scheme gives superior performance over the existing one.

References

[1] A. Burchard, J. Liebeherr, Y. Oh, and S. H. Son. Assigning Real-Time Tasks to Homogeneous Multiprocessor Systems. Technical Report CS-94-01, University of Virginia, Computer Science Department, January 1994.

[2] S. Davari and S. K. Dhall. An On Line Algorithm for Real-Time Allocation. In *IEEE Real-Time Systems Symposium*, pages 194–200, 1986.

[3] S. K. Dhall and C. L. Liu. On a real-time scheduling problem. *Operations Research*, 26(1):127–140, January/February 1978.

[4] J. D. Gafford. Rate-Monotomic Scheduling. *IEEE Micro*, pages 34–39, June 1991.

[5] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst Case Performance Bounds for Simple One-dimensional Packing Algorithms. *SIAM Journal of Computing*, 3:299–325, 1974.

[6] J. Y.-T. Leung and J. Whitehead. On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks. *Performance Evaluation*, 2:237–250, 1982.

[7] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment. *Journal of the ACM*, 20(1):46–61, January 1973.

[8] Y. Oh and S. H. Son. On-line Task Allocation Algorithms for Hard Real-Time Multiprocessor Systems. Submitted for Publication.

[9] L. Sha and J.B. Goodenough. Real-time Scheduling Theory and Ada. *Computer*, pages 53–66, April 1990.

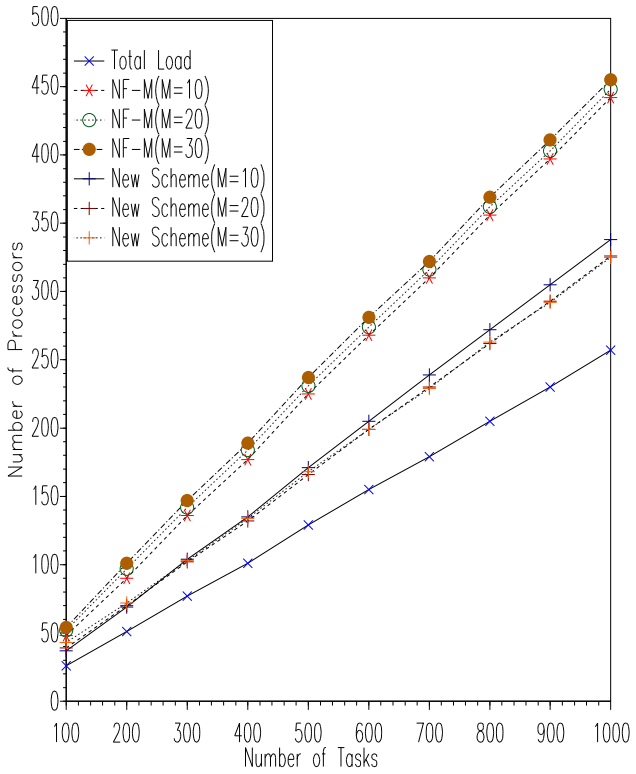

Figure 1: Task Sets with $\alpha = 0.5$.