Assigning Real-Time Tasks to Homogeneous Multiprocessor Systems

Almut Burchard *
Jörg Liebeherr **
Yingfeng Oh **
Sang H. Son **

* School of Mathematics
Georgia Institute of Technology
Atlanta, GA 30332

** Computer Science Department
University of Virginia
Charlottesville, VA 22903

January 6, 1994

Abstract

Optimal scheduling of real-time tasks on multiprocessor systems is known to be computationally intractable for large task sets. Any practical scheduling algorithm for assigning real-time tasks to a multiprocessor system presents a trade-off between its computational complexity and its performance. The performance of a scheduling algorithm is measured in terms of the additional number of processors required to arrive at a schedule without deadline violations as compared to an optimal algorithm. In this study, new schedulability conditions are presented for homogeneous multiprocessor systems where individual processors execute the rate-monotonic scheduling algorithm. The conditions are used to develop new strategies for assigning real-time tasks to processors. The performance of the new strategies is shown to be significantly better than suggested by the existing literature. Under the (realistic) assumption that the load of each real-time task is small compared to the processing speed of each processor, it is shown that all processors can be almost fully utilized. Task assignment strategies are proposed for scenarios where the task set is known *a priori* (*off-line schemes*), and where the real-time task set can change dynamically (*on-line schemes*).

Key Words: Hard Real-Time Systems, Multiprocessor Systems, Rate-Monotonic Scheduling, Periodic Tasks, Task Assignment Scheme.

1 Introduction

The distinguishing feature of real-time computer systems is their attempt to achieve both logical and temporal correctness of computations. A computation is temporally correct if it finishes within a specified time frame. In this sense, all time-constrained computer applications require a real-time computer system. Commonly, however, real-time computer systems are used if violations of temporal correctness may result in drastic consequences as, for example, in power plants, hospitals, or manufacturing and transportation systems.

In most real-time applications, the computer system is subject to arrivals of messages containing monitor and control information from many different sources. These messages can arrive at any time, however, the minimal distance between consecutive arrivals from the same source is constrained. Each message arrival initiates the request for executing a computational task. The task must be completed before the arrival of the subsequent message from the same source. Thus, the earliest arrival time of the next message from the same source is the deadline for executing the task. In the worst case, each task is requested periodically, where the period is given by the minimum time interval between consecutive message arrivals from a particular source. We refer to tasks that are requested at most periodically and must finish execution before the end of the next period as *(periodic) real-time* tasks.

To maximize the number of real-time tasks that can be processed without timing violations, real-time computer systems use sophisticated scheduling algorithms to decide the order in which tasks are executed. The performance of a scheduling algorithm is measured by its ability to generate a feasible schedule for a set of real-time tasks. A schedule for assigning tasks to one or more processors is said to be *feasible* if the execution of each task can be completed before its deadline. A feasible schedule is said to be *minimal* if there is no feasible schedule utilizing less processors. A scheduling algorithm is said to be *optimal* if for any set of tasks the algorithm finds a minimal schedule.

Scheduling algorithms can be divided into *fixed priority* and *dynamic priority* algorithms. In fixed priority algorithms, the priority of a task remains constant at all times, whereas in dynamic priority algorithms, the priority of a task may change during its execution.

In their seminal work, Liu and Layland [8] showed that in single processor systems the dynamic priority *earliest-deadline-due* (EDD) algorithm which assigns the highest priority to the task closest to the end of its period is optimal among all scheduling algorithms. They also showed that the *rate-monotonic* (RM) algorithm which assigns higher priority to tasks with shorter periods is optimal among all fixed priority scheduling algorithms. For both the EDD and RM algorithms, Liu and Layland derived sufficient conditions under which the respective algorithms yield feasible schedules.

Such conditions are referred to as *schedulability conditions*. Recently, necessary and sufficient schedulability conditions were stated for both the RM algorithm [6] and the EDD algorithm [12].

Due to its low computational overhead the RM algorithm is widely regarded as an appropriate algorithm for scheduling real-time tasks on uniprocessor systems. Recently proposed extensions to the RM algorithm have increased its practical relevance [10, 11].

Even though real-time computer systems are expected to greatly benefit from multiprocessor technology, employing multiprocessor systems for real-time applications has shown to be difficult. A major obstacle is that scheduling algorithms for real-time multiprocessor systems are significantly more complex than for uniprocessor systems. In multiprocessor systems, the scheduling algorithm must not only specify an ordering of tasks, but also must determine the specific processor to be used. Leung and Whitehead [7] proved that finding a minimal schedule for a given set of real-time tasks in a multiprocessor system is NP-hard. Therefore, research efforts have focused on the development of suitable heuristic algorithms which can be efficiently implemented, yet, require only a limited number of additional processors as compared to an optimal algorithm.

There are two strategies for scheduling real-time tasks on a multiprocessor system. In a *global* scheme each occurrence of a real-time task may be executed on a different processor. In contrast, a *partitioning* scheme enforces that all occurrences of a particular task are executed on the same processor. A partitioning scheme has several advantages over a global scheme. First, partitioning schemes are less complex since the overhead of multiprocessor scheduling merely consists in assigning tasks to processors. Note that the assignment is performed only once for each task, i.e., before the task is executed for the first time. Secondly, if the assignment of tasks to processors is completed, well-known uniprocessor scheduling algorithms can be used for each processor.

The performance of an partitioning scheme is determined by two factors; the *task assignment algorithm* which distributes tasks to the processors, and the scheduling algorithm which determines the order of task executions on each processor. For a given scheduling algorithm, an *optimal* task assignment algorithm achieves a feasible schedule for each processor with the least number of processors. However, the problem of finding an optimal assignment of tasks to processors for fixed priority scheduling algorithms, in particular the RM algorithm, as well as for dynamic priority scheduling algorithms, in particular EDD, was shown to be NP-hard [7].

In this study, we are concerned with task assignment schemes for homogeneous multiprocessor systems where each processor executes the RM scheduling algorithm. This problem has been addressed in a number of studies [1, 2, 4, 9]. Typically, the task assignment schemes apply variants of well-known heuristic bin-packing algorithms where the set of processors is regarded as a set of bins [1].

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

The decision whether a processor is full is determined by a schedulability condition. All existing task assignment schemes are based on the sufficient schedulability conditions for uniprocessor systems derived in [8] and variants of this condition [3]. Thus, the existing assignment schemes differ mainly in the choice of the bin-packing heuristic.

In [4], two heuristic assignment schemes are proposed, referred to as Rate-Monotonic Next-Fit (RMNF) and Rate-Monotonic First-Fit (RMFF). The schemes are based on the *next-fit* and *first-fit* bin-packing heuristic, respectively. In both schemes, tasks are sorted in decreasing order of their periods before the assignment is started. Tasks are assigned to a so-called *current* processor until the schedulability condition is violated, in which case the current processor is marked full and a new processor is selected. RMFF first tries to accommodate a task in a processor marked as full before assigning it to the current processor. The First-Fit Decreasing-Utilization Factor (FFDUF) method is a variation of the first-fit heuristic scheme. Here, tasks are sorted in the order of their load factor [1]. In [9], a *best-fit* bin-packing heuristic is used as the basis for the Rate-Monotonic Best-Fit (RMBF) scheme. Similarly to RMFF, RMBF attempts to assign tasks to processors that have been marked as full. However, in RMBF, the full processors are inspected in a specific order. As in [4], tasks are assumed to be sorted by their period.

Since the above schemes require that the entire task set is known before starting the task assignment they are referred to as *off-line* schemes. In contrast, *on-line* schemes allow the task set to change dynamically, that is, tasks can be added to or deleted from the task set. On-line task assignment schemes can be implemented with lower computational complexity than off-line schemes, but may require more processors for a feasible schedule. An on-line task assignment scheme based on the next-fit bin-packing heuristics and referred to as Next-Fit-M is described in [2]. In Next-Fit-M, M is a parameter denoting the maximal number of processors which is considered for assigning a new task.

In all studies, 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. Unfortunately, bounds for the existing schemes are only available as asymptotic bounds, that is, as $\lim_{N_{opt}\to\infty} N/N_{opt}$.

In Table 1(a), we summarize the heuristic methods from the literature with their performance bounds. The measure $O(K)$ denotes the upper bound of the computational complexity for scheduling a set of K real-time tasks.

Our approach for developing task assignment schemes 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 conditions that allow to assign more tasks to each

Scheme	Asymptotic Upper Bound: $\lim_{N_{opt}\to\infty} N/N_{opt}$	Complexity	Type
RMNF [4]	2.67	$O(K \log K)$	*off-line*
RMFF [4]	2.33	$O(K \log K)$	*off-line*
FFDUF [1]	2	$O(K \log K)$	*off-line*
RMBF [9]	2.33	$O(K \log K)$	*off-line*
Next-Fit-M [2]	$2.28+O(1/M)$	$O(K)$	*on-line*

(a) Existing Task Assignment Schemes.

RMST	$1/(1-\alpha)$	$O(K \log K)$	*off-line*
RMGT	1.75	$O(K \log K)$	*off-line*
RMGT/M	$1/(1-\alpha)+O(1/M)$	$O(K)$	*on-line*

(b) Proposed Task Assignment Schemes.

Table 1: Comparison of Task Assignment Schemes.

processor. We show that the maximum achievable load on each processor is significantly higher than suggested by previous work. If the load factor of each task is small compared to the processing power of a processor – a very realistic assumption considering the state-of-the-art of hardware technology – we will show that each processor can be almost fully utilized. More precisely, the *Rate-Monotonic Small-Tasks (RMST) scheme* proposed in this study achieves an asymptotic bound of $\lim_{N_{opt}\to\infty} N/N_{opt} \leq 1/(1-\alpha)$, where α is the maximal load factor of an individual task. For general task sets we propose the *Rate-Monotonic General-Tasks (RMGT) scheme* which yields an asymptotic bound of $\lim_{N_{opt}\to\infty} N/N_{opt} \leq 1.75$. Different from previous work we also derive bounds of the performance parameter N/N_{opt} for $N < \infty$. In addition to the *off-line* schemes RMST and RMGT we propose an *on-line* task assignment scheme for general task sets. The scheme is referred to as *Rate-Monotonic General-Tasks/M (RMGT/M) scheme*, where M is a parameter denoting the number of processors to which a new task can be assigned. In Table 1(b) we summarize the performance characteristics of the assignment schemes proposed in this study.

The remainder of this study is structured as follows. In Section 2 we present our model for

real-time tasks and multiprocessor systems. In Section 3 we derive a tight schedulability condition for the RM scheduling algorithm in a uniprocessor system that improves on the results presented in [8]. We also prove a scheduling result for multiprocessor systems which can be interpreted as *dual* result to our uniprocessor scheduling condition. In Section 4 we construct two simple assignment schemes, referred to as *RMST* and *RMGT*. With our theoretical results from Section 3 we can prove bounds for the number of processors required with these schemes. The on-line assignment scheme RMGT/M is presented in Section 5. Again, we use the results from Section 3 to derive performance bounds. In Section 6, we conclude the study with a short discussion of our results.

2 Model Description

We assume that the real-time computer system consists of a homogeneous 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 also 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 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

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

  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. 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.

- Throughout this paper, we assume that the *rate-monotonic* (RM) algorithm is used to schedule tasks on each processor. That is, task τ_i has precedence over task τ_j, if $T_i < T_j$. We assume that scheduling of tasks is preemptive, and that task execution can be resumed without loss after interruptions.

3 Schedulability Conditions

In this section we derive two sufficient schedulability conditions for processors which schedule tasks with the RM algorithm. The first result, presented in Theorem 1, is a simple modification of the result for uniprocessor systems by Liu and Layland [8]. Our result yields a higher utilization of the processor if the task periods satisfy certain constraints. On uniprocessor system, Theorem 1 does not provide a significant improvement for scheduling real-time tasks. For multiprocessor scheduling, however, we can divide a large tasks set into subsets in such a way that we can make use of the sharpened condition on all but possibly one processor.

In our second result, stated in Theorem 2, we present a schedulability condition for the RM algorithm in multiprocessor systems. Theorem 2 uses exactly the same parameters, i.e., the total load U and the number of tasks K, as the uniprocessor result by Liu and Layland [8]. In fact, Theorem 2 can be interpreted as as a dual result to the schedulability conditions given in [8] for multiprocessor system. Both results coincide for the special case $K = 2$.

A result similar to our Theorem 2 was conjectured in [4], but not proven. A partial proof, yet incomplete and needing additional assumptions was given in [3].

3.1 Rate-Monotonic Scheduling in Uniprocessor Systems

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.

Theorem 1 *Given a real-time task set $\tau_1, \ldots, \tau_K$. Define*

$$S_i := \log_2 T_i - \lfloor \log_2 T_i \rfloor \qquad i = 1, \ldots, K \tag{1}$$

and

$$\beta := \max_{1 \le i \le K} S_i \; - \; \min_{1 \le i \le K} S_i \tag{2}$$

(a) *If $\beta < 1 - 1/K$, and the total load satisfies*

$$U \le (K-1)\left(2^{\beta/(K-1)} - 1\right) + 2^{1-\beta} - 1 \tag{3}$$

then the task set is schedulable on one processor with the RM algorithm.

(b) *If $\beta \ge 1 - 1/K$, and the total load satisfies*

$$U \le K\left(2^{1/K} - 1\right) \tag{4}$$

then the task set is schedulable on one processor with the RM algorithm.

Both conditions are tight.

Note that Inequality (4) is exactly the schedulability condition given by Liu and Layland [8]. Theorem 1 improves upon [8] when $\beta < 1 - 1/K$, since the strict convexity of the function $f(x) = x(2^{1/x} - 1)$ implies that

$$(K-1)\left(2^{\beta/(K-1)} - 1\right) + 2^{1-\beta} - 1 > K(2^{1/K} - 1) \ . \tag{5}$$

Throughout the paper, we will use a simpler version of Theorem 1. The simplified schedulability condition is given in the following corollary.

Corollary 1 *Given a set of real-time tasks $\tau_1, \ldots, \tau_K$, and define β as in (2). If the total load satisfies*

$$U \leq \max\{\ln 2, 1 - \beta \ln 2\} \tag{6}$$

then the task set can be scheduled on one processor.

Proof. Because both schedulability conditions (3) and (4) of Theorem 1 are strictly decreasing with respect to K, we have that

$$K(2^{1/K} - 1) > \lim_{K \to \infty} K(2^{1/K} - 1) = \ln 2 \tag{7}$$

and

$$\begin{aligned} (K-1)\left(2^{\beta/(K-1)} - 1\right) + 2^{1-\beta} - 1 &> \lim_{K \to \infty} (K-1)\left(2^{\beta/(K-1)} - 1\right) + 2^{1-\beta} - 1 && (8) \\ &= \beta \ln 2 + 2^{1-\beta} - 1 && (9) \\ &> 1 - \beta \ln 2 && (10) \end{aligned}$$

Schedulability now follows from Theorem 1. □

The remainder of this subsection contains the proof of Theorem 1. For the proof, we will need three lemmas. Lemma 1, due to Lehoczky [6], gives the necessary and sufficient schedulability for the RM algorithm in a uniprocessor system.

Lemma 1 *Given a set of real-time tasks $\tau_1, \tau_2, \ldots, \tau_K$. Assume the tasks are ordered with increasing period, $T_1 \leq \cdots \leq T_K$. Then, a task τ_k always meets its deadline T_k under rate-monotonic scheduling, if and only if there exists a time $0 < t \leq T_k$ such that*

$$t \geq \sum_{i=1}^{k} \left\lceil \frac{t}{T_i} \right\rceil C_i \tag{11}$$

We will need the following special cases of Lemma 1. If $T_K \leq 2T_1$, then condition (11) reduces to

$$\exists j \leq k: \quad T_j \geq \sum_{i=1}^{j-1} 2C_j \; + \; \sum_{i=j}^{k} C_i \; . \tag{12}$$

If the task set consists of only two tasks, (11) reduces to

$$\left\lfloor \frac{T_2}{T_1} \right\rfloor (T_1 - C_1) \geq C_2 \quad \text{or} \quad T_2 \geq \left\lceil \frac{T_2}{T_1} \right\rceil C_1 + C_2 \tag{13}$$

The next lemma states that the RM algorithm is distinguished by a special property, which also holds for EDD, however, not for any other fixed-priority or dynamic scheduling algorithm. The proof is a simple application of Lemma 1, but the result is surprisingly powerful. We will apply Lemma 2 to obtain sufficient schedulability conditions for general task sets from schedulability conditions for task sets where the longest period is at most twice as long as the shortest period. Lemma 2 implies that it is not necessary to assume that a task set is ordered by periods in order to apply the schedulability conditions in [4, 9], a fact overlooked in both references.

Lemma 2 *Given a task set $\tau_1, \tau_2, \ldots, \tau_K$, and a task $\tau = (C, T)$ with $T \leq T_i$ for $i = 1, \ldots, K$. If τ and $\tau_1, \ldots, \tau_K$ cannot be scheduled together on one processor with the RM scheduling algorithm, then also $(2C, 2T), \tau_1, \ldots, \tau_K$ cannot be scheduled.*

Proof. We assume that the tasks are ordered such that $T_1 \leq \cdots \leq T_K$. Denote by k the smallest index, such that (C, T) together with $(C_1, T_1), \ldots (C_k, T_k)$ is not schedulable.

There are two possible cases. If $2T > T_k$, then we can use schedulability condition (12) for both task sets in question. But clearly, the condition

$$2C \; + \; \sum_{i=1}^{j-1} 2C_i \; + \; \sum_{i=j}^{k} C_i \; > \; T_j \tag{14}$$

for all $j \leq k$ is equivalent to

$$\sum_{i=1}^{j-1} 2C_i \; + \; \sum_{i=j}^{k} C_i + 2C \; > \; T_j \tag{15}$$

for all j. Also

$$C \; + \; \sum_{i=1}^{k} C_i \; > \; T \tag{16}$$

is equivalent to

$$\sum_{i=1}^{k} 2C_i \; + 2C \; > \; 2T \tag{17}$$

which shows the claim that $(2C, 2T), (C_1, T_1), \ldots, (C_k, T_k)$ cannot be scheduled. If, on the other hand, $2T \leq T_k$, then Lemma 1 implies that for all $0 \leq t \leq T_k$

$$t < \left\lceil \frac{t}{T} \right\rceil C + \sum_{i=1}^{k} \left\lceil \frac{t}{T_i} \right\rceil C_i \tag{18}$$

$$\leq \left\lceil \frac{t}{2T} \right\rceil 2C + \sum_{i=1}^{k} \left\lceil \frac{t}{T_i} \right\rceil C_i \tag{19}$$

Again we have shown that $(2C, 2T), (C_1, T_1), \ldots, (C_k, T_k)$ is not schedulable. □

Lemma 3 is a corollary of Lemma 2 which applies to multiprocessor systems.

Lemma 3 *Assume that the task set $\tau_1, \ldots, \tau_K$ cannot be scheduled on N processors. Then the task set $\tau'_1, \ldots, \tau'_K$ given by*

$$C'_i = U_i T'_i, \qquad T'_i = 2^{S_i} \tag{20}$$

cannot be scheduled on N processors.

Proof. For all tasks τ_i with $T_i > 2^{S_i}$ we replace (C_i, T_i) by $(2^{-m}C_i, 2^{-m}T_i)$, where m is selected such that $2^{-m}T_i < 2^{S_i}$. It is easy to see that by scaling t in the Lehoczky schedulability conditions (11), the replacement does not change the schedulability of the task set. Also, the values for U_i and S_i remain unchanged. So, we may assume that $T_i \leq 2^{S_i}$ for all i. If $T_i = 2^{S_i}$ for all i, we are done. Otherwise, we select τ_k such that $T_k = \min_i(T_i)$ and replace (C_k, T_k) by $(2C_k, 2T_k)$. Clearly, this does not change the load factor U_k. Lemma 2 implies that the resulting task set cannot be scheduled on N processors. We repeat this procedure until we arrive at a task set with $T_i = 2^{S_i}$ for all tasks. □

Proof of Theorem 1. We will show that any set of K tasks that cannot be scheduled on a single processor violates condition (3), if $\beta < 1 - 1/K$, and violates condition (4), if $\beta \geq 1 - 1/K$. To show that the bounds from Theorem 1 are tight we will construct a task set that cannot be scheduled on one processor, but whose total load is arbitrarily close to the bounds in (3) or (4). The proof will proceed in four steps.

- **(1)** Formulate Theorem 1 in terms of a problem of minimizing U as a function of its variables $\underline{C} = (C_1, C_2, \ldots, C_K)$ and $\underline{T} = (T_1, T_2, \ldots, T_K)$.
- **(2)** Fix the periods $\underline{T} = (T_1, T_2, \ldots, T_K)$ and minimize U over the execution times $\underline{C} = (C_1, C_2, \ldots, C_K)$. Use the result to express the execution times as functions of the periods.
- **(3)** Transform the reduced minimization problem into a convex problem.
- **(4)** Solve the convex minimization problem.

(1) Assume that the task set $(C_1, T_1), \ldots, (C_K, T_K)$ cannot be scheduled on one processor. Since the conditions given by (3) and (4) are strictly decreasing with K, they are certainly violated for a task set if they are violated for a subset. Hence, we may assume without loss of generality that all proper subsets of the task set *can* be scheduled on one processor. By Lemma 3 we can assume that

$$T_1 \leq \ldots \leq T_K \leq 2^\beta T_1 \tag{21}$$

Since, by assumption, the proper subset $\tau_1, \ldots, \tau_{K-1}$ can , but the complete task set $\tau_1, \ldots, \tau_K$ cannot be scheduled on a single processor, task τ_K, which has the lowest priority, misses its deadline. By the schedulability condition in (12), this is equivalent to

$$\sum_{i=1}^{j-1} 2C_i + \sum_{i=j}^{K} C_i > T_j \qquad\qquad j = 1, \ldots, K \tag{22}$$

We will minimize the total load U as a function of the execution times and the periods of all tasks. Thus, we have to solve the following problem.

$$\textbf{minimize} \quad U(\underline{C}, \underline{T}) = \sum_{i=1}^{K} \frac{C_i}{T_i} \tag{23}$$

subject to

$$\sum_{i=1}^{j-1} 2C_i + \sum_{i=j}^{K} C_i \geq T_j \qquad j = 1, \ldots, K \tag{24}$$

$$0 \leq C_i \leq T_i \qquad\qquad i = 1, \ldots, K \tag{25}$$

$$T_1 \leq \cdots \leq T_K \leq 2^\beta T_1 \qquad \beta \leq 1 \tag{26}$$

We replaced "$>$" by "$\geq$" in (24) and (25) to ensure that the minimum is attained at some point. Since the functional is continuous, this does not affect the minimal value of U. Note that $U(\underline{C}, \underline{T})$ in (23) is not a convex function of its arguments. Hence, standard (nonlinear) optimization methods cannot be applied.

(2) We will show that $U(\underline{C}, \underline{T})$ takes its minimal value in a point where conditions (24) holds with equality. Suppose that we have found the minimum, say U^*, for the objective function with $\underline{C}^* = (C_1^*, C_2^*, \ldots, C_K^*)$, and $\underline{T}^* = (T_1^*, T_2^*, \ldots, T_K^*)$. If for some $j > 1$, inequality (24) is strict, we set

$$\tilde{C}_i = \begin{cases} C_{j-1}^* - \varepsilon & \text{if } i = j-1 \\ C_j^* + \varepsilon & \text{if } i = j \\ C_i^* & \text{otherwise} \end{cases} \tag{27}$$

where ε is defined by

$$\varepsilon := \sum_{i=1}^{j-1} 2C_i^* + \sum_{i=j}^{K} C_i^* - T_j > 0 \tag{28}$$

Then side condition (24) is unchanged for $i \neq j$, and holds with equality for $i = j$. The total load at this point satisfies

$$U(\underline{\tilde{C}}, \underline{T}^*) = U^* - \varepsilon \left(\frac{1}{T^*_{j-1}} - \frac{1}{T^*_j} \right) \leq U^* \tag{29}$$

where we have used inequality (21). So we found a new minimum of the functional.

Similarly, if there is strict inequality in condition (24) for $j = 1$, we set

$$\tilde{C}_i = \begin{cases} C^*_1 + \varepsilon & \text{if } i = 1 \\ C^*_K - 2\varepsilon & \text{if } i = K \\ C^*_i & \text{otherwise} \end{cases} \tag{30}$$

where

$$\varepsilon := \sum_{i=1}^{K} C^*_i - T^*_1 \ > 0 \tag{31}$$

The total load satisfies

$$U(\underline{\tilde{C}}, \underline{T}^*) = U^* - \varepsilon \left(\frac{2}{T^*_n} - \frac{1}{T^*_1} \right) \leq U^* \ , \tag{32}$$

where we have used inequality (21). Again we found a new minimum of the functional.

Summarizing, we have shown that U takes its minimum in a point with

$$\sum_{i=1}^{j-1} 2C^*_i + \sum_{i=j}^{K} C^*_i = T^*_j \qquad \text{for } j = 1, \ldots, K \tag{33}$$

Subtracting equations in (33) for consecutive indices, and subtracting (33) for $j = 1$ from (33) for $j = K$, we obtain the following identities

$$C^*_j = T^*_{j+1} - T^*_j \qquad \text{for } j = 1, \ldots, K-1 \tag{34}$$

$$C^*_K = 2T^*_1 - T^*_K \tag{35}$$

Note that the side conditions in (25) are satisfied automatically. Thus, we have reduced the problem to

$$\textbf{minimize} \quad U(\underline{T}) = \sum_{i=1}^{K-1} \frac{T_{i+1} - T_i}{T_i} \ + \ \frac{2T_1 - T_K}{T_K} \tag{36}$$

subject to

$$T_1 \leq \cdots \leq T_K \leq 2^\beta T_1 \tag{37}$$

(3) Substituting

$$x_i := \log_2 \frac{T_{i+1}}{T_i} \qquad i = 1, \ldots, K-1 \tag{38}$$

$$x_K := \log_2 \frac{2T_1}{T_K} \tag{39}$$

we rewrite the minimization problem in (36) and (37) as

$$\textbf{minimize} \quad U(\underline{x}) = \sum_{i=1}^{K} (2^{x_i} - 1) \tag{40}$$

subject to

$$x_i \geq 0, \qquad i = 1, \ldots, K \tag{41}$$

$$x_K \geq 1 - \beta \tag{42}$$

$$\sum_{i=1}^{K} x_i = 1 \tag{43}$$

Relation (43) is a consequence of the definitions in (38) and in (39).

(4) The minimization problem in (40) - (43) is a (strictly) convex problem, since the functional is a sum of convex functions and the side conditions describe a convex set. At this point, the minimization can be completed with a Lagrange multiplier method.

More directly, it follows from the strict convexity of the problem that there is a unique critical point which is the absolute minimum. This critical point must be symmetric under any symmetry of the problem. If we disregard condition (42), the problem is completely symmetric under permutation of the indices. Hence, in the minimum, all x_i^* must be equal, and side condition (43) demands that

$$x_i^* = 1/K, \qquad \text{for } i = 1, \ldots, K \tag{44}$$

Note that for $\beta \geq 1 - 1/K$, condition (42) is satisfied automatically. Reversing the transformations in (34) – (35) and (38) – (39), we obtain the following solution of the original problem in (23) – (26).

$$C_i^* = a\, 2^{i/K}(2^{1/K} - 1), \qquad T_i^* = a\, 2^{i/K}, \qquad \text{for } i = 1, \ldots, K \tag{45}$$

where $a > 0$ is any number. On this task set, $U(\underline{C}, \underline{T})$ takes the minimal value

$$U^* = K\left(2^{1/K} - 1\right) \tag{46}$$

given on the right hand side of (4). Note that the task set given in (45) can be scheduled on a single processor. However, if any of the execution times C_i^* is replaced by $\tilde{C}_i > C_i^*$, then the resulting task set is not schedulable.

If $\beta \leq 1 - 1/K$, the minimum satisfies

$$x_K^* = 1 - \beta \tag{47}$$

Problem (40) – (43) is symmetric under permutation of $x_1, \ldots, x_{K-1}$. Hence, in the minimum, all values for x_i $(i \neq K)$ must be equal. From (43) and (47) we obtain the solution

$$x_i^* = \frac{\beta}{K-1} \qquad i = 1, \ldots, K-1 \tag{48}$$

Transforming back with equations (34) – (35) and (38) – (39) gives the final solution

$$C_i^* = a\, 2^{\beta i/(K-1)}(2^{\beta/(K-1)} - 1) \qquad T_i^* = a\, 2^{\beta i/(K-1)} \qquad i = 1, \ldots, K-1 \tag{49}$$

$$C_K^* = a\, 2^{\beta K/(K-1)}(2^{1-\beta} - 1) \qquad T_K^* = a\, 2^{\beta K/(K-1)} \tag{50}$$

where $a > 0$ is any number. For this task set, $U(\underline{C}, \underline{T})$ takes the minimal value

$$U^* = (K-1)\left(2^{\beta/(K-1)} - 1\right) + 2^{1-\beta} - 1\ , \tag{51}$$

The task set given in (49) and (50) can be scheduled on one processor. But if any of the execution times C_i^* is replaced by $\tilde{C}_i > C_i^*$, then the resulting task set cannot be scheduled. This completes the proof. □

3.2 Rate-Monotonic Scheduling in Multiprocessor Systems

In the previous subsection we were concerned with scheduling real-time tasks on a uniprocessor system that employs the RM scheduling algorithm. Theorem 1 addresses the question: When can a set of tasks be scheduled on *one* processor? For the answer we found the worst case task set which can still be scheduled on a single processor. For the corresponding result in multiprocessor systems we have to find the minimal number of processors that is needed to find a feasible assignment of a task set to a multiprocessor system. Obviously, in the worst case, only one task can be assigned to each processor. Therefore, we have to be concerned with the question: When can a set of tasks be scheduled on less than one processor for *each* task?

The answer to this question is given in Theorem 2. Roughly, Theorem 2 says, that if K processors are needed to schedule a set of K tasks, then the load on each processor cannot be much less than 1/2.

Theorem 2 *If the total load of a set of K real-time tasks satisfies*

$$U \leq \frac{K}{2^{1/K} + 1} \tag{52}$$

then the task set can be scheduled with the RM algorithm on less than K processors. The condition is tight.

The following example shows that Theorem 2 is a true multiprocessor result, and cannot be obtained as a corollary of Theorem 1. We select an an integer $K \geq 3$, and set

$$C_i = 2^{i/K} \qquad T_i = 2^{i/K}(2^{1/K}+1) \qquad \text{for } i = 1, \ldots, K \tag{53}$$

Since $U = \frac{K}{2^{1/K}+1}$ the task set can be scheduled on less than K processors by Theorem 2. However, since $1/(2^{1/K}+1) > \sqrt{2}-1$ the schedulability conditions in Theorem 1 fail for any pair of tasks. As a result, assignment schemes based on Theorems 2 can achieve a better processor utilization than schemes based on uniprocessor results alone.

The following corollary provides a good approximation of Theorem 2. The statement is asymptotically exact for $K \to \infty$, and never differs by more than 0.07 from the exact value.

Corollary 2 *If the total load U of a set of K real-time tasks satisfies*

$$U \leq \frac{K}{2} - \frac{\ln 2}{4} \tag{54}$$

then the task set can be scheduled with the RM algorithm on less than K processors.

Proof. The proof follows from the following inequalities.

$$\frac{K}{2} - \frac{\ln 2}{4} < \frac{K}{2^{1/K}+1} \leq \frac{K}{2} - \frac{1}{6} \tag{55}$$

□

Next we present the proof of Theorem 2. For the proof we will need a technical lemma, given in Lemma 4. We will use the lemma to show that for the task set which satisfies the inequality of Theorem 2 with the lowest value for the total load, the execution times C_i satisfy the same ordering relations $C_1 \leq \cdots \leq C_k \leq 2C_1$ as the periods.

Lemma 4 *Assume that a set of K nonnegative numbers $C_1, \ldots, C_K$ has the following property. For every $1 \leq j \leq K$ there exists at least one index i such that*

$$C_j = m_i := \min\{2C_1, \ldots, 2C_{i-1}, C_{i+1}, \ldots, C_K\} \qquad \text{or} \qquad 2C_j = m_i \tag{56}$$

For such a set, either

$$C_1 \leq \cdots \leq C_K \leq 2C_1 \tag{57}$$

or there exists an index $1 \leq j \leq K$ and numbers $a > b \geq 0$ such that

$$C_i = \begin{cases} a & \text{if } i < j \\ b & \text{if } i = j \\ 2a & \text{if } i > j \end{cases} \tag{58}$$

Proof. For $K \leq 2$ the statement is trivially correct. For $K > 2$, consider a task set that satisfies the assumption but violates equation (57). We have to distinguish two cases. Either, $C_{j-1} > C_j$ for some index $1 < j \leq K$, or $C_K > 2C_1$. We will only prove the first case, since the proof for the second case is completely analogous.

Assume that there exists an index $1 < j \leq K$ with $C_{j-1} > C_j$. Then we obtain by definition of m_i, that for all $i < j - 1$,

$$m_i \leq C_j < C_{j-1} \tag{59}$$

and for all $i > j$,

$$m_i \leq 2\,C_j < 2\,C_{j-1} \tag{60}$$

But with the assumption in (56) we must have that $m_j = 2C_{j-1}$. By definition of m_j, this means

$$2C_j < 2C_{j-1} \leq \begin{cases} 2C_i & \text{if } i < j-1 \\ C_i & \text{if } i > j \end{cases} \tag{61}$$

Using (56), we have

$$m_i = \begin{cases} 2C_j & \text{if } i < j \\ 2C_{j-1} & \text{if } i = j \\ C_j & \text{if } i > j \end{cases} \tag{62}$$

It follows that (58) holds with $a = C_{j-1}$ and $b = C_j$. □

Proof of Theorem 2. We want to find the smallest value of U for a set of K tasks that cannot be scheduled on less than K processors. Since for $K = 1$ there is nothing to show, we will assume $K \geq 2$. The proof will proceed in five steps.

- **(1)** Formulate Theorem 2 in terms of a problem of minimizing U as a function of its variables (C_i, T_i).
- **(2)** Fix the execution times $\underline{C} = (C_1, C_2, \ldots, C_K)$ and minimize U over the periods $\underline{T} = (T_1, T_2, \ldots, T_K)$. Use the result to express the periods as functions of the execution times.
- **(3)** Transform the minimization problem into a convex minimization problem.
- **(4)** Solve the minimization problem with standard methods.

(1) Assume that the tasks $(C_1, T_1), \ldots, (C_K, T_K)$ cannot be scheduled on less than K processors. We want to show that the total load U violates (52). With Lemma 3 we can assume that

$$T_1 \leq \cdots \leq T_K \leq 2T_1 \tag{63}$$

The assumption that the K tasks cannot be scheduled on less than K processors is equivalent to the statement that no two tasks can be scheduled on one processor. By (13) this is equivalent to

$$\begin{cases} T_i < C_i + C_j & \text{if } i < j \\ T_i < C_i + 2C_j & \text{if } i > j \end{cases} \tag{64}$$

So we have to solve the following minimization problem.

$$\textbf{minimize} \quad U(\underline{C}, \underline{T}) = \sum_{i=1}^{K} \frac{C_i}{T_i} \tag{65}$$

subject to

$$T_i < C_i + C_j \qquad i < j \tag{66}$$

$$T_i < C_i + 2C_j \qquad i > j \tag{67}$$

$$0 \leq C_i \leq T_i \qquad i = 1, \ldots, K \tag{68}$$

$$T_1 \leq \cdots \leq T_k \leq 2T_1 \tag{69}$$

As in the proof of Theorem 1, we replace "$<$" by "$\leq$" to enforce that the minimum is attained at some point.

(2) Since the partial derivatives of $U(\underline{C}, \underline{T})$

$$\frac{\partial U}{\partial T_i}(\underline{C}, \underline{T}) = -\frac{C_i}{T_i^2} < 0 \tag{70}$$

are negative, $U(\underline{C}, \underline{T})$ is minimal if we we choose the T_i as large as possible. We enforce the first two side conditions, (66) and (67), by setting

$$T_i = C_i + \min\{2C_1, \ldots, 2C_{i-1}, C_{i+1}, \ldots, C_k\} \tag{71}$$

To simplify (71), we will show that in the absolute minimum of $U(\underline{C}, \underline{T})$, conclusion (57) of Lemma 4 holds. To this end, we first have to show that the assumptions of Lemma 4 hold. Assume that (56) is not satisfied, that is, for some index j there exists neither $i < j$ such that $m_i = C_j$, nor $i > j$ such that $m_i = 2C_j$. With (71), the objective function can be phrased exclusively in terms of $\underline{C}$. Since the total derivative

$$\frac{dU}{dC_j}(\underline{C}) = \frac{m_j}{(C_j + m_j)^2} > 0 \tag{72}$$

is positive, we can lower the value of the functional $U(\underline{C})$ by lowering the values for C_j. Thus, condition (56) of Lemma 3 is satisfied in any critical point of the functional.

Next we show that the second outcome of Lemma 4 is not feasible in a minimum of the functional. If the task execution times satisfy (58) then

$$C_i = \begin{cases} a & \text{if } i < j \\ b & \text{if } i = j \\ 2a & \text{if } i > j \end{cases} \tag{73}$$

with $0 \leq b < a$. At such a point the periods of the tasks are given by

$$T_i = \begin{cases} a+b & \text{if } i < j \\ b+2a & \text{if } i = j \\ 2b+2a & \text{if } i > j \end{cases} \tag{74}$$

We obtain for the total load factor

$$U(\underline{C},\underline{T}) = (K-1)\frac{a}{a+b} + \frac{b}{2a+b} \tag{75}$$

Setting

$$t = \frac{b}{a} \tag{76}$$

equation (75) reads

$$U(t) = (K-1)\frac{1}{1+t} + \frac{t}{2+t} \tag{77}$$

which is nonincreasing with t. Consequently, for $0 \leq t \leq 1$, U assumes its minimum at $t = 1$, that is $a = b$. Hence, condition (57) holds.

Since we showed that in a minimum the C_i satisfy condition (57), we obtain with (71) that

$$T_i = C_i + C_{i+1} \qquad i = 1, \ldots, K-1 \tag{78}$$

$$T_K = C_K + 2C_1 \tag{79}$$

Note that the side condition in (69) is always satisfied.

Summarizing, we have reduced the problem to

$$\textbf{minimize} \quad U(\underline{C}) = \sum_{i=1}^{K-1} \frac{C_i}{C_i + C_{i+1}} + \frac{C_K}{C_K + 2C_1} \tag{80}$$

subject to

$$C_i \geq 0 \qquad i = 1, \ldots, K \tag{81}$$

(3) We perform a transformation of variables. Define

$$\begin{aligned} x_i &:= \log_2 \tfrac{C_{i+1}}{C_i} && \text{if } i < K \\ x_K &:= \log_2 \tfrac{2C_1}{C_K} \end{aligned} \tag{82}$$

Then the optimization problem of (80) and (81) reads

$$\textbf{minimize} \quad U(\underline{x}) = \sum_{i=1}^{K} \frac{1}{1+2^{x_i}} \tag{83}$$

subject to

$$x_i \geq 0 \qquad i = 1, \ldots, K \tag{84}$$

$$\sum_{i=1}^{K} x_i = 1 \tag{85}$$

where equation (85) is a consequence of definition (82).

(4) Since the optimization problem in (83) - (85) is strictly convex, and since both the functional and the side conditions are symmetric under permutation of the indices, the unique minimum is also symmetric under permutations of indices. From the side condition in (85), we directly obtain that the solution must be

$$x_i^* = 1/K \qquad i = 1, \ldots, K \tag{86}$$

Transforming back to the original variables we obtain the following solution in terms of execution times and periods:

$$C_i^* = a\, 2^{i/K} \tag{87}$$

$$T_i^* = a\, 2^{i/K}(2^{1/K}+1) \tag{88}$$

where $a > 0$ can be any number. For this task set, U has the minimal value

$$U^* = K/(2^{1/K}+1) \tag{89}$$

The task set with parameters as in (87) and (88) can be scheduled on less than K processors. But if all C_i^* are replaced by $\tilde{C}_i > C_i^*$, then the resulting task can only be scheduled on K processors. This completes the proof. □

4 Assignment Schemes for Multiprocessor Systems

In this section, we show how to use our theoretical results from Section 3 to design a new class of off-line assignment schemes for distributing a set of real-time tasks to a set of processors. We will compare our assignment schemes with an optimal scheme which always utilizes the minimum number of processors. For a given task set, we denote by N_{opt} the number of processors needed by an optimal assignment scheme.

We will propose two assignment schemes. The first scheme, referred to as *Rate-Monotonic Small-Task* or *RMST* scheme, is intended for task sets where the load factor U_i of each real-time task is small compared to the processing speed of each processor. The second scheme, referred to as *Rate-Monotonic General-Task* or *RMGT* scheme, applies to general task sets.

Previously proposed assignment schemes only consider the load factors of the tasks [1, 2, 4, 9]. Our schemes gain superiority by additionally taking into account the task periods. Before we present the schemes, let us review the maximum performance that can be achieved with a task assignment scheme using information on the load factors only. In this case, the sufficient schedulability conditions given in [8] and it variants [4, 9] are the best available schedulability conditions. If these conditions are used then the load assigned to any pair of processors exceeds $\ln 2$. So, if N processors are used, then

$$U > \frac{\ln 2}{2} N \tag{90}$$

This bound cannot be improved beyond $(\sqrt{2} - 1)N$. For example, a set with K real-time tasks where $U_i = \sqrt{2} - 1 + \varepsilon$ for each task cannot be scheduled onto K processors with this condition. Thus, all task assignment schemes that use the scheduling condition of [8] are strictly limited in the performance they can achieve.

Similar arguments show that if the load factor of every task is less than α, then a next-fit task assignment scheme based on [8] will ensure that the load on all but one processor is at least $\ln 2 - \alpha$. This shows that

$$U > (\ln 2 - \alpha)N \tag{91}$$

For $\alpha < 1/2$ this bound is best possible. More sophisticated bin-packing heuristics, such as first-fit or best-fit, can improve the average performance a lot, but one can construct task sets such that $U \leq N \ln 2$. Consequently, in the worst case, $N/N_{opt} \geq 1/\ln 2$.

A moment's consideration shows, that the bounds given in inequalities in (90) and (91) are far below the bounds that can be achieved with an optimal assignment scheme. By Theorem 2, any set of K real-time tasks with $U_i \leq 1/2 - \varepsilon$ for all tasks can be scheduled on less than K processors if the number of tasks is sufficiently large. And we can certainly assign task sets with small load factors

to processors in such a way that the load on all but one processor exceeds 1/2. This argument suggests that the best bound for an optimal assignment scheme is of the form

$$U > \frac{N}{2} - const. \tag{92}$$

Similarly, if the load factor of every task is bounded above by α, one might hope to prove that

$$U > (1 - \alpha)N - const. \tag{93}$$

Note that these inequalities have the same form as corresponding inequalities for periodic tasks *without* deadline constraints. In particular, the leading terms $N/2$ and $(1 - \alpha)N$ are best possible.

The RMST and RMGT assignment schemes proposed here create task assignments that satisfy bounds of the form (93) and (92). In the following we discuss both schemes and prove their properties.

4.1 RMST – An Assignment Scheme for Small Tasks

We first consider the problem of scheduling a set of tasks with small load factors. Denote by

$$\alpha := \max_{i=1,\ldots,K} U_i \tag{94}$$

the maximal load factor of any single real-time task. For all practical purposes, we may assume that a task set contains only "small" tasks if $\alpha \leq 1/2$.

Recall that by Corollary 1 the minimal achievable load on a single processor is larger than $1 - \beta \ln 2$ where β is defined as in (2). The main idea of RMST is to partition the tasks in such a way that on each processor, β has a small value.

It is convenient to visualize the values of S_i for a given task set as points on a circle with circumference one. Starting at any point on the circle and proceeding clockwise, we assign tasks to processors, using the schedulability condition of Theorem 1. Then, the value of β at each processor is given by the length of the arc spanned by the tasks that are assigned to that processor.

The RMST scheme is summarized in Algorithm 1. It can be easily verified that the computational complexity of Algorithm 1 is determined by the sorting of tasks in Step 1. Thus, the complexity of RMST is $O(K \log K)$ where K is the size of the task set.

(1) Order the task set such that $0 \leq S_1 \leq \cdots \leq S_K < 1$ and set $S_{K+1} := S_1 + 1$. Set the task index to $i := 1$, and the processor index to $n = 0$.

(2) Select an empty processor with index $n := n + 1$. Assign task τ_i to processor n, that is $\rho_n := U_i$. Set $S := S_i$, and $\tilde{\beta}_n := 0$.

(3) Increase the task index, $i := i + 1$, and set $\tilde{\beta}_n := S_i - S$. If the schedulability condition (6)

$$U_i + \rho_n \leq \max\{\ln 2, 1 - \tilde{\beta} \ln 2\}$$

is satisfied, assign task τ_i to processor n by setting $\rho_n := \rho_n + U_i$, and continue with step **(3)**. Otherwise, continue with step **(2)**.

(4) When all tasks have been assigned set $\tilde{\beta}_n := S_{K+1} - S$ and terminate.

Algorithm 1. Rate-Monotonic Small Task (RMST).

Let us illustrate the RMST scheme with an example. In Table 1 we show the parameters for a set of 10 tasks, also including the values of S_i. The task set is already ordered according to S_i. Note that for the parameter set in Table 1 we obtain $\alpha = 0.3167$. In Figure 1, we depict the values for S_i on a circle with circumference one. The shaded areas indicate the assignment of tasks to processors as obtained by RMST. The areas are labeled with the value of β as defined in (2) for a particular processor [2]. Thus, three processors are required to schedule the set of tasks of Table 1, where tasks $\tau_1, \tau_2, \tau_3, \tau_4$ are assigned to the first processor, τ_5, τ_6, τ_7 to the second processor, and $\tau_8, \tau_9, \tau_{10}$ to the third processor. Note that any assignment scheme proposed in the literature would utilize four processors.

Next we will show that the RMST scheme satisfies performance bounds of the form given in (92).

Theorem 3 *For any given task set $\tau_1, \ldots, \tau_K$, the RMST scheme arrives at a feasible assignment of tasks to processors. If the maximal load factor of any single task is*

$$\alpha := \max_{i=1,\ldots K} \frac{C_i}{T_i} < 1 \tag{95}$$

then the number N of processors needed by RMST satisfies the inequality

$$N < \frac{U}{1-\alpha} + 2 - \frac{1 - \ln 2}{1 - \alpha} \ . \tag{96}$$

[2] Note that β is slightly different from $\tilde{\beta}_n$ in Algorithm 1; $\tilde{\beta}_n$ also contains the value of S_i of the first task τ_i *not* assigned to processor n.

$i =$	1	2	3	4	5	6	7	8	9	10
$T_i =$	65	280	36	150	20	45	400	7	230	60
$C_i =$	16	27	11	31	3	14	113	2	70	19
$U_i =$	0.2461	0.964	0.3055	0.2067	0.15	0.3111	0.2825	0.2857	0.3043	0.3167
$S_i =$	0.0223	0.1292	0.1699	0.2288	0.3219	0.4918	0.6438	0.8073	0.8454	0.9068

Figure 1: Example Parameter Set.

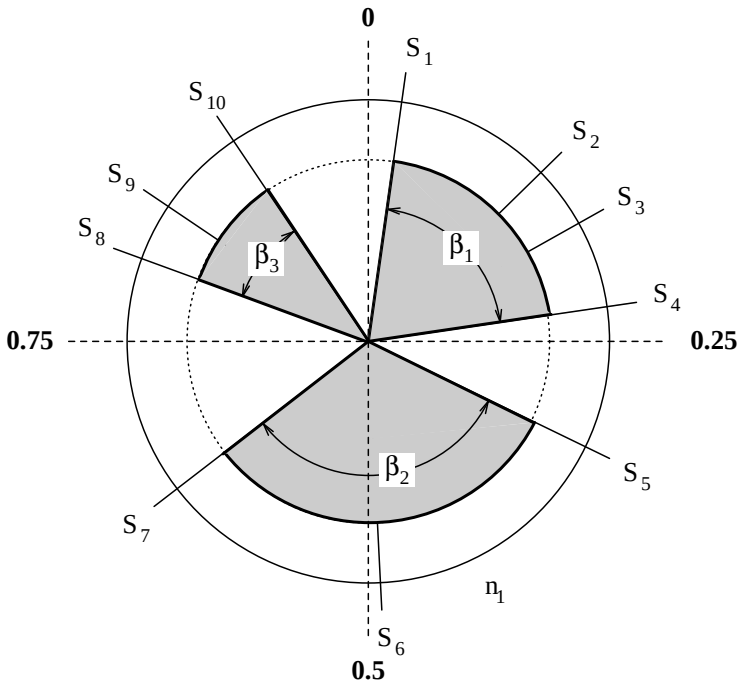

Figure 2: Processor Assignment for Algorithm 1.

The inequality in (96) is not tight, however, one can construct task sets for which RMST requires at least $U/(1-\alpha)$ processors. For $\alpha \leq 1/2$, the bounds obtained with the RMST scheme improve upon any previously proposed assignment scheme. For $\alpha > 1/2$, the RMGT scheme presented in the next subsection will give better bounds than RMST. Since in most practical real time systems the load imposed by individual real-time tasks is small compared to the power of the processors, the result of Theorem 3 shows to be highly relevant for practical real-time applications.

Proof of Theorem 3. Denote by K_n the number of tasks assigned to the n-th processor, and by ρ_n the load on the processor due to these tasks. Define $\tilde{\beta}_n$ as $\tilde{\beta}_n = S_{i_{n+1}} - S_{i_n}$, where i_n is the index of the first task assigned to processor n. We also set $S_{i_{N+1}} = 1 + S_1$. Note that these

definitions are identical to the values of $\tilde{\beta}_n$ after Algorithm 1 has terminated.
The task assignment resulting from Algorithm 1 is feasible by Corollary 1. We now estimate N, the number of processors needed by Algorithm 1 to schedule a given task set. By our schedulability condition in (6), the loads on the processors satisfy

$$\rho_n > 1 - \tilde{\beta}_n \ln 2 - \alpha \qquad n = 1, \ldots N-1 \tag{97}$$

$$\rho_{N-1} + \rho_N > 1 - \tilde{\beta}_{N-1} \ln 2 \tag{98}$$

Adding (97) and (98), and using that by construction

$$\sum_{n=1}^{N} \tilde{\beta}_n = 1 \tag{99}$$

we obtain

$$U = \sum_{n=1}^{N} \rho_N \tag{100}$$

$$> \sum_{n=1}^{N-2} (1 - \tilde{\beta}_n \ln 2 - \alpha) + 1 - \tilde{\beta}_{N-1} \ln 2 \tag{101}$$

$$\geq (N-2)(1-\alpha) + 1 - \ln 2 \tag{102}$$

Solving for N yields assertion (96). □

In the following corollaries we compare the number of processors required to schedule a task set using RMST with the number of processors required for an optimal assignment scheme. Corollary 3 gives the bound of the performance parameter N/N_{opt} for any number N of processors. In Corollary 4 we present the asymptotic limit which is included in Table 1.

Corollary 3 *For any task set, the number N of processors needed by RMST satisfies*

$$\frac{N}{N_{opt}} < \frac{1}{1-\alpha} + \frac{2}{N_{opt}} \tag{103}$$

N_{opt} denotes the number of processors utilized by an optimal assignment scheme.

Proof. For any schedule, we must have $N_{opt} \geq U$. With (96) we obtain

$$N < \frac{N_{opt}}{1-\alpha} + 2 - \frac{1 - \ln 2}{1-\alpha} \tag{104}$$

Dividing by N_{opt} gives the claim. □

Corollary 4 *Let $\{\tau_i \mid i = 1, 2, \ldots\}$ be a given infinite task set. Assume that the load factor of any single task does not exceed α. Denote by $U(k)$ the total load factor of tasks $\tau_1, \ldots, \tau_k$. Denote by $N(k)$ the number of processors used by RMST to schedule $\tau_1, \ldots, \tau_k$, and by $N_{opt}(k)$ the number of processors needed in an optimal assignment. If*

$$\lim_{k \to \infty} U(k) = \infty \tag{105}$$

then

$$\lim_{k \to \infty} \frac{U(k)}{N(k)} \geq 1 - \alpha \tag{106}$$

$$\lim_{k \to \infty} \frac{N(k)}{N_{opt}(k)} \leq \frac{1}{1 - \alpha} \tag{107}$$

The bounds are tight.

Proof. We obtain (106) by dividing by N in equation (96) of Theorem 3 and passing to the limit. For equation (107) we pass to the limit in inequality (103) of Corollary 3. □

4.2 RMGT – An Assignment Scheme for General Task Sets

The RMST scheme from the previous subsection provides excellent bounds for task sets where the maximal load factor of the tasks is limited by $U_i \leq 1/2$. Next we propose a simple task assignment scheme, referred to as *Rate-Monotonic General-Task (RMGT)* scheme which is applicable to unrestricted task sets. We show that RMGT is able to find a feasible task assignment with less than $N < 2U + 2$ processors.

For the RMGT scheme we partition the task set into two groups, such that the load factors of tasks in the first and second group, respectively, satisfy $U_i \leq 1/3$ and $U_i > 1/3$. Tasks in the first group are assigned to processors with the RMST scheme in Algorithm 1. Tasks from the second group are assigned to processors with a first-fit heuristic. The heuristic assigns at most two tasks to one processor using the exact schedulability condition from (13). The complete RMGT scheme is summarized in Algorithm 2.

Partitioning the task set in Algorithm 2 involves a computational complexity of $O(K)$ where K is the total number of tasks. Recall from subsection 4.1 that Algorithm 1 has a complexity of $O(|\mathcal{G}_1| \log |\mathcal{G}_1|)$. Since a first fit bin-packing algorithm for tasks in $\mathcal{G}_2$ can be implemented with a complexity of $O(|\mathcal{G}_2| \log |\mathcal{G}_1|)$ [5], the worst case computational complexity of the RMGT scheme is given by $O(K \log K)$.

In the following theorem, Theorem 4, we show that the number of processors needed by RMGT for scheduling an arbitrary set of real-time tasks satisfies a bound similar to (92). In Theorem 5 and Corollary 5 we prove bounds for the performance of the RMGT scheme.

(1) Partition the set of tasks into two groups:

$$\mathcal{G}_1 = \{\tau_i \mid U_i \leq 1/3\}$$
$$\mathcal{G}_2 = \{\tau_i \mid U_i > 1/3\}$$

(2) Use the RMST scheme in Algorithm 1 to assign the task set $\mathcal{G}_1$.

(3) Assign tasks in $\mathcal{G}_2$ as follows:

(3.1) Set the task index to $i := 1$ and assign τ_1 to an empty processor with index 1. Set $\rho_1 := U_1$.

(3.2) Increase the task index to $i := i + 1$ and consider task τ_i. Use a first-fit heuristic to find a processor n that contains a task τ_j such that

$$\left\lfloor \frac{T_j}{T_i} \right\rfloor (T_j - C_i) \geq C_j \quad \text{or} \quad T_j \geq \left\lceil \frac{T_j}{T_i} \right\rceil C_i + C_j$$

if $T_i < T_j$, and

$$\left\lfloor \frac{T_i}{T_j} \right\rfloor (T_i - C_j) \geq C_i \quad \text{or} \quad T_i \geq \left\lceil \frac{T_i}{T_j} \right\rceil C_j + C_i$$

if $T_i \geq T_j$. If such a processor exists, assign task τ_i to processor n by setting $\rho_n := \rho_n + U_i$, otherwise, assign task τ_i to an empty processor with index m, and set $\rho_m := U_i$. Continue with step **(3.2)**.

(3.3) Terminate when all tasks in $\mathcal{G}_2$ have been assigned.

Algorithm 2. Rate-Monotonic General Task (RMGT).

Theorem 4 *For any task set $\tau_1, \ldots, \tau_K$, the RMGT algorithm arrives at a feasible processor assignment. The number N of processors utilized by RMGT to schedule the task set satisfies*

$$N < 2U + \frac{5}{2}\ln 2 - \frac{1}{3} \approx 2U + 1.42 \tag{108}$$

Proof. Given a task set $\tau_1, \tau_2, \ldots, \tau_K$. With Theorem 3 and Lemma 1, RMGT arrives at a feasible processor assignment. Next we estimate the number of processors used by *RMGT*. Assume that Algorithm 2 assigns to n_1 processors each two tasks from $\mathcal{G}_2$, and to n_2 processors each one task from $\mathcal{G}_2$. Also assume that Algorithm 1 utilizes n_3 processors for tasks in $\mathcal{G}_1$. Denote the total load factors assigned to the three groups of processors by $U^{(1)}$, $U^{(2)}$, and $U^{(3)}$, respectively. Then we obtain:

$$U^{(1)} > \frac{2n_1}{3} \geq \frac{n_1}{2} \tag{109}$$

$$U^{(2)} > \frac{n_2}{2} - \frac{\ln 2}{4} \tag{110}$$

$$U^{(3)} > \frac{2n_3}{3} - \frac{1}{3} - \ln 2 \geq \frac{n_3}{2} - \ln 2 + \frac{1}{6} \tag{111}$$

Inequality (109) holds, because each processor in this group has been assigned two tasks, each with a load larger than 1/3. Inequality (110) follows from Corollary 2. Inequality (111) follows from Theorem 3 provided that $n_3 \geq 3$. Because the joint load on any two processors exceeds $\ln 2$ by construction, (111) holds also if $n_3 < 3$. Adding the three inequalities and solving for N yields the claim in (108). □

Theorem 5 *The number of processors utilized by* RMGT *to assign a task set satisfies*

$$\frac{N}{N_{opt}} \leq \frac{7}{4} + \frac{15\ln 2 - 4}{8N_{opt}} \approx \frac{7}{4} + \frac{0.79}{N_{opt}} \tag{112}$$

where N_{opt} is the number of processors utilized by an optimal assignment. The inequality becomes false, if the right hand side is replaced by anything less than 7/4.

Proof. As in the proof of Theorem 4, we denote by n_2 the number of processors that have been assigned one task from $\mathcal{G}_2$. Then, from inequalities (109), (110), and (111), we obtain

$$N_{opt} \geq U > \frac{2}{3}(N - n_2) + \frac{n_2}{2} - \frac{1}{3} - \ln 2 - \frac{\ln 2}{4} \tag{113}$$

$$= \frac{2}{3}N - \frac{n_2}{6} - \frac{1}{3} - \frac{5}{4}\ln 2 \tag{114}$$

Moreover, since we used exact schedulability conditions to schedule class $\mathcal{G}_2$, we have that

$$N_{opt} \geq n_2 \tag{115}$$

We obtain

$$\begin{aligned} N_{opt} &\geq \min_{0 \leq n \leq N} \max\{U, n_2\} && (116)\\ &\geq \frac{4}{7}N + \frac{2}{7} - \frac{15}{14}\ln 2 && (117) \end{aligned}$$

which proves (112).

To show that the inequality in (112) is tight we will construct a task set such that $N/N_{opt} = 7/4$. The task set consists of $13m$ ($m > 0$) tasks. For the construction of the tasks we select two small positive numbers ε and δ with

$$\varepsilon \ln 2 \leq \delta < \frac{3}{2}\varepsilon \ln 2 \tag{118}$$

We label the tasks by $\tau_{i,j}$, where $1 \leq i \leq m$, and $1 \leq j \leq 13$. The tasks are given by

$$C_{i,j} = U_{i,j}T_{i,j} \ , \qquad T_{i,j} = 2^{S_{i,j}} \tag{119}$$

Then the values for $U_{i,j}$ and $S_{i,j}$ are given by

$$U_{i,j} = \begin{cases} 1/2 - \delta & \text{if } j = 2,5,8,11 \\ 1/3 & \text{if } j = 1,6,10 \\ 1/6 - \delta & \text{if } j = 3,4,7,9,12,13 \end{cases} \tag{120}$$

and

$$S_{i,j} = \begin{cases} (12i + j)\varepsilon & \text{if } 1 \leq j \leq 12 \\ (12i + 11)\varepsilon & \text{if } j = 13 \end{cases} \tag{121}$$

An optimal assignment will require exactly $4m$ processors to schedule the given task set. In particular, the optimal assignment distributes the following groups of tasks to one processor each.

$$\{\tau_{i,1}, \tau_{i,2}, \tau_{i,3}\} \ , \ \{\tau_{i,4}, \tau_{i,5}, \tau_{i,6}\} \ , \ \{\tau_{i,7}, \tau_{i,8}, \tau_{i,9}, \tau_{i,13}\} \ , \ \{\tau_{i,10}, \tau_{i,11}, \tau_{i,12}\} \quad \text{for } i = 1, \ldots, m \ . \tag{122}$$

Since the total load of the task set is given by $U \geq (1 - 4\delta)(4m)$, the given assignment is optimal, if ε and δ are chosen sufficiently small. The assignment is feasible by Corollary 1.

RMGT as given in Algorithm 2 first divides the task set into two groups. In the first group we have for each i

$$\mathcal{G}_1 = \{\tau_{i,j} \mid j = 1,3,4,7,9,10,12,13;\ i = 1, \ldots\ m\} \tag{123}$$

These tasks are assigned with Algorithm 1, which results in the following processor assignment.

$$\{\tau_{i,1}, \tau_{i,3}, \tau_{i,4}\} \ , \quad \{\tau_{i,6}, \tau_{i,7}, \tau_{i,9}\} \ , \quad \{\tau_{i,10}, \tau_{i,13}, \tau_{i,12}\} \quad \text{for } i = 1, \ldots, m \tag{124}$$

The second group is given by

$$\mathcal{G}_2 = \{\tau_{i,j} \mid j = 2, 5, 8, 11;\ i = 1, \ldots\ m\} \tag{125}$$

The load factor of each task in $\mathcal{G}_2$ is given by $1/2 - \delta$. Thus, according to equation (13), no two tasks from $\mathcal{G}_2$ can be scheduled on the same processor. So, RMGT needs $7m$ processors for the same task set. □

Corollary 5 *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 tasks $\tau_1, \ldots, \tau_k$. Denote by $N(k)$ the number of processors used by* RMGT *to schedule $\tau_1, \ldots, \tau_k$, and by $N_{opt}(k)$ the number needed in an optimal assignment. If*

$$\lim_{k\to\infty} U(k) = \infty \tag{126}$$

then

$$\lim_{k\to\infty} \frac{\rho(k)}{N(k)} \geq \frac{1}{2} \tag{127}$$

$$\lim_{k\to\infty} \frac{N(k)}{N_{opt}(k)} \leq \frac{7}{4} = 1.75 \tag{128}$$

Both bounds are tight.

Proof. Inequality (127) is obtained by dividing (108) of Theorem 4 by $2N$, and passing to the limit. Inequality (128) follows by passing to the limit in (112) of Theorem 5. □

5 RMGT/M – An On-Line Task Assignment Scheme

In this section we propose an on-line version of the task assignment schemes presented in section 4. Recall that on-line task assignment schemes do not require that the entire task set is known *a priori*. Rather, on-line schemes provide procedures for dynamically adding new tasks and deleting existing tasks at any time. We assume that the number of processors in the real-time computer system is not constrained. Thus, new tasks can always be added to the processor system.

We refer to the new scheme as *Rate-Monotonic General-Tasks/M (RMGT/M) scheme*, where M is a parameter denoting the number of processors to which a new task can be assigned. RMGT/M has the following properties. If a task is dynamically added to a processor, the assignments of existing tasks remain unchanged. However, if a task is deleted from a particular processor, possibly all tasks on this processor are moved to other processors.

The RMGT/M scheme is based on the schedulability conditions used for the off-line RMST scheme discussed from subsection 4.1. In RMGT/M, each task is assigned to one of a fixed number of M classes. The class membership, say m, of a task τ is determined by the following expression:

$$m = \left\lfloor M(\log_2(T) - \lfloor \log_2(T) \rfloor) \right\rfloor + 1 \tag{129}$$

Each processor is assigned tasks from only one class. Thus, at each processor the value of β as defined in (2) is bounded above by $1 - \ln 2/M$. For each class, the RMGT/M scheme keeps one so-called *current processor*. If a new task from class m is added to the task set, RMGT/M first attempts to accommodate the task to the current processor for class m. A complete description of the procedures for adding and deleting a task $\tau = (C, T)$ is given in Algorithm 3.

In Algorithm 3, adding and deleting of a task $\tau = (C, T)$ are performed by procedures *AddTask()* and *DeleteTask()*. *AddTask()* first determines the class membership, say m, of the new task τ (Step 1). 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 (Steps 5 and 6). If the load factor of τ is large, no other task will be assigned to this processor (Steps 8 and 9).

Similarly to *AddTask()*, procedure *DeleteTask()* first determines the class membership of the task to be deleted (Step 1). If the task, say τ, is assigned to the current processor of its class, it is merely canceled (Step 3). Otherwise, all tasks at this processor (except τ_i) are assigned to different processors with procedure *AddTask()* (Steps 5 and 6), and the processor is labeled as empty (Step 8).

The performance bounds of the RMGT/M scheme are given in Theorem 6 and Corollary 6. Corollary 6 states the asymptotic bound of RMGT/M. The bounds given in Table 1 can be obtained from the corollary.

Theorem 6 *If a dynamically changing task set is scheduled with RMGT/M then the number of processors needed satisfies*

$$N < \frac{U}{1 - \ln 2/M - \alpha} + M \tag{130}$$

The bound is tight if $\alpha \leq (1 - \ln 2/M)/2$. *We also have*

$$N < \frac{2U}{1 - \ln 2/M} + M \tag{131}$$

which is a tight bound if $\alpha \geq (1 - \ln 2/M)/2$.

Global functions:

curr(m)	–	Returns the current processor for class m.
proc(τ)	–	Returns the processor index assigned to task τ.
newproc()	–	Returns the index of an empty processor.
empty(n)	–	Labels processor n as empty processor.

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

```
        begin
1.        m := ⌊M (log2 (T) − ⌊log2 (T)⌋)⌋ + 1;
2.        if (ρ_curr(m) + C/T ≤ 1 − ln 2/M) then
3.            ρ_curr(m) := ρ_curr(m) + C/T;
4.        else if (ρ_curr(m) < C/T) then
5.            curr(m) := newproc();
6.            ρ_curr(m) := C/T;
7.        else
8.            x := newproc();
9.            ρ_x := C/T;
10.       endif
        end
```

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

```
        begin
1.        m := ⌊M(log2 (T) − ⌊log2 (T)⌋)⌋ + 1;
2.        if (proc(τ) = curr(m)) then
3.            ρ_proc(τ) := ρ_proc(τ) − C/T;
4.        else
5.            for each {τ̃ | proc(τ̃) = proc(τ), τ̃ ≠ τ} do
6.                AddTask(τ̃);
7.            endfor
8.            empty(proc(τ));
9.        endif
        end
```

Algorithm 3. Rate-Monotonic General Task/M (RMGT/M).

Proof. The schedulability condition used for Algorithm 3 in Step 2 of procedure *AddTask()* enforces that at any instant, the load on all processors but the M current processors exceeds both $1-\ln 2/M-\alpha$ and $(1-\ln 2/M)/2$. □

Corollary 6 *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 3, and by $N_{opt}(k)$ the number of processors used by an optimal scheme. If*

$$\lim_{k\to\infty} U(k) = \infty \tag{132}$$

then we have the asymptotic bounds

$$\lim_{k\to\infty} \frac{U(k)}{N_M(k)} \geq \max\left\{1 - \ln 2/M - \alpha \ , \ \frac{1-\ln 2/M}{2}\right\} \tag{133}$$

$$\lim_{k\to\infty} \frac{N_M(k)}{N_{opt}(k)} \leq \min\left\{\frac{1}{1-\ln 2/M-\alpha} \ , \ \frac{2}{1-\ln 2/M}\right\} \tag{134}$$

The bounds are tight.

Proof. We obtain both (133) and (134) by passing to the limit in (130) and (131). □

From the derived bounds we see that the performance of RMGT/M is sensitive to the selection of M, the number of task classes. The asymptotic bounds in (133) and (134) 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 (131). Since the right hand side of (131) is a strictly convex function of M, we can calculate the unique minimum which is denoted by M^*:

$$M^* = \sqrt{2U\ln 2} + \ln 2 \tag{135}$$

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

$$\frac{U}{N} \geq (1 - \ln 2/M - \alpha)(1 - M/N) = \frac{1}{2} - O(1/\sqrt{U}) \tag{136}$$

and hence

$$\frac{N}{N_{opt}} \leq 2 + O(1/\sqrt{U}) \tag{137}$$

Similarly, if $\alpha < 1/2$, we can minimize the right hand side of (130) over M and obtain that the optimal choice for M should be as close as possible to

$$M^* = \frac{\sqrt{U\ln 2} + \ln 2}{1-\alpha} \tag{138}$$

If we choose $M \sim \sqrt{U}$, we obtain with (130) the following bound for the average utilization at each processor.

$$\frac{U}{N} \geq (1 - \ln 2/M - \alpha)(1 - M/N) = 1 - \alpha - O(1/\sqrt{U}) \ . \tag{139}$$

and N/N_{opt} is given by

$$\frac{N}{N_{opt}} \leq \frac{1}{1-\alpha} + O(1/\sqrt{U}) \tag{140}$$

6 Conclusions

We derived new schedulability conditions for scheduling (periodic) real-time tasks on uniprocessor and multiprocessor systems. Each processor was assumed to use the rate-monotonic scheduling algorithm. Based on the schedulability conditions we developed three assignment schemes, called *RMST*, *RMGT*, and *RMGT/M*, for assigning tasks to the processors. Both *RMST* and *RMGT* are so-called *off-line* task assignment schemes, that is, the entire set of real-time tasks is assumed to be known. The *on-line* scheme *RMGT/M* allows that real-time tasks are dynamically added to or deleted from the task set.

For each of the schemes, we obtained upper bounds for the performance parameter N/N_{opt}, where N is the number of processors required to schedule a task set with RMST or RMGT, and N_{opt} is the number of processors needed in an optimal assignment. We also obtained lower bounds for the average processor utilization. We provided asymptotic limits of the bounds. For RMST, the asymptotic bound for N/N_{opt} was proven to be $1/(1-\alpha)$ where α is the maximal load factor of the tasks in the given task set. For the RMGT scheme we proved an asymptotic bound of 7/4. In RMGT/M, the asymptotic bound was shown to be at most $1/(1-\alpha) + O(1/M)$, where M is a parameter.

The improvement of the performance bounds compared to previous existing results were achieved with rather simple assignment algorithms. The strength of the presented schemes resulted from novel schedulability conditions. We conjecture that both the RMST and RMGT scheme leave ample space for improvements. For example, the following modifications to our assignment schemes may result in better performance bounds.

- For RMST one can use tighter schedulability conditions than the one we applied. Without additional computational cost one could use the following more precise condition which can be obtained from Theorem 1.

$$U_j + \rho_n \leq \begin{cases} \beta \ln 2 + 2^{1-\beta} - 1 & \text{if } \beta \leq 1/2 \\ \ln 2 & \text{if } \beta > 1/2 \end{cases}$$

This will lower the number of processors used, however, the asymptotic limit for N/N_{opt} will not change. RMST can be further improved by using the exact schedulability condition in Theorem 1 which contains the number of tasks, K, as a parameter. Then the schedulability condition in Algorithm 1 takes the form

$$U_j + \rho_n \leq \begin{cases} (K-1)(2^{\beta/(K-1)} - 1) + 2^{1-\beta} - 1 & \text{if } \beta \leq 1 - 1/K \\ \ln 2 & \text{if } \beta > 1 - 1/K \end{cases}$$

- Note that RMGT never assigns 'large' tasks (with $U_i > 1/3$) and 'small' tasks (with $U_i \leq 1/3$) to the same processor. However, it is very likely, that processors that are assigned 'large' tasks can accommodate some additional 'small' tasks. This will increase the average load per processor significantly, and may also improve the worst case bound for N/N_{opt}.

References

[1] S. Davari and S. K. Dhall. An On Line Algorithm for Real-Time Allocation. In *19th Annual Hawaii International Conference on System Sciences*, pages 194–200, 1986.

[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. *Scheduling Periodic-Time-Critical Jobs on Single Processor and Multiprocessor Computing Systems*. PhD thesis, University of Illinois at Urbana-Champaign, 1977.

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

[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. P. Lehoczky, L. Sha, and Y. Ding. The Rate-Monotonic Scheduling Algorithm: Exact Characterization and Average Behavior. In *IEEE Real-Time Systems Symposium*, pages 166–171, 1989.

[7] 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.

[8] 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.

[9] Y. Oh and S. H. Son. Tight Performance Bounds of Heuristics for a Real-Time Scheduling Problem. Submitted for Publication.

[10] L. Sha, J. P. Lehoczky, and R. Rajkumar. Solutions for Some Practical Problems in Prioritized Preemptive Scheduling. In *IEEE Real-Time Systems Symposium*, pages 181–191, 1986.

[11] J. A. Stankovic and K. Ramamritham (editors). *Hard Real-Time Systems*. IEEE Computer Society Press, 1988.

[12] Q. Z. Zheng and K. G. Shin. On the Ability of Establishing Real-Time Channels in Point-to-Point Packet Switched Networks. to appear: IEEE Transactions on Communications.

A Average-Case Performance Evaluation of RMST and RMGT

In this study, the performance bounds of the new assignment schemes, RMST and RMGT, were derived under worst-case assumptions. While a worst-case analysis assures that the performance bounds are satisfied for any task set, it does not provide insight into the average behavior of the assignment schemes. To obtain the average-case performance of the RMST and RMGT schemes, one can analyze the schemes with probabilistic assumptions, or conduct simulation experiments to empirically study the average performance. Since a probabilistic analysis of our algorithms is beyond the scope of this study, we resort to simulation to gain insight into the average-case behavior of RMST and RMGT.

We present simulation experiments for large task sets with $100 \leq K \leq 1000$ tasks. In each experiment, we vary the value of parameter $\alpha = \max_{i=1,\ldots,K} U_i$, the maximal load factor of any task in the set. 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 $1 \leq C_i \leq \alpha T_i$. The performance metric in all experiments is the number of processors required to assign a given task set. We compare the RMST and RMGT schemes with several existing assignment schemes. All assignment schemes are executed on identical task sets. The following off-line assignment schemes are considered:

- Rate-Monotonic Small-Tasks (RMST) (section 4.1),
- Rate-Monotonic General-Tasks (RMGT) (section 4.2),
- Rate-Monotonic-Next-Fit (RMNF) [4],
- Rate-Monotonic-First-Fit (RMFF) [4],
- Rate-Monotonic-Best-Fit (RMBF) [9] [3]

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 outcome of the simulation experiments is shown in Figures A.1 – A.3. The maximum load of a task is set to $\alpha = 0.2$ in Figure A.1, to $\alpha = 0.5$ in Figure A.2, and to $\alpha = 0.8$ in Figure A.3. In the Figures, each data point depicts the average value of 15 independently generated task sets with identical parameters. Note that the RMGT scheme gives the best performance in all experiments. In Figure A.1, the performance of RMST and RMGT cannot be distinguished since RMST and RMGT are identical if $\alpha < 0.3$. As we increase the value of α, we observe that the performance of RMST decreases.

The graphs in Figures Figures A.1 – A.3 show that in all schemes, the number of processors required for the respective schemes, increases proportionally to the total load. In Table A.1 we

[3] In all simulations, the results obtained with RMFF and RMBF were identical.

show the range of values of

$$N/U := \frac{\text{Number of Processors Used}}{\text{Total Load}}$$

as obtained for the different assignment schemes in all task sets. The data in Table A.1 was collected from the same simulation experiments used for Figures A.1 – A.3.

N/U	RMNF	RMFF	RMST	RMGT
$\alpha = 0.2$	[1.42, 1.55]	[1.30, 1.45]	[1.06, 1.20]	[1.06, 1.20]
$\alpha = 0.5$	[1.50, 1.64]	[1.29, 1.36]	[1.15, 1.26]	[1.14, 1.22]
$\alpha = 0.8$	[1.50, 1.67]	[1.27, 1.38]	[1.30, 1.44)	[1.18, 1.33]

Table A.1: Simulation Results for N/U.
[a,b] denotes that $a \leq N/U \leq b$ in all simulations.

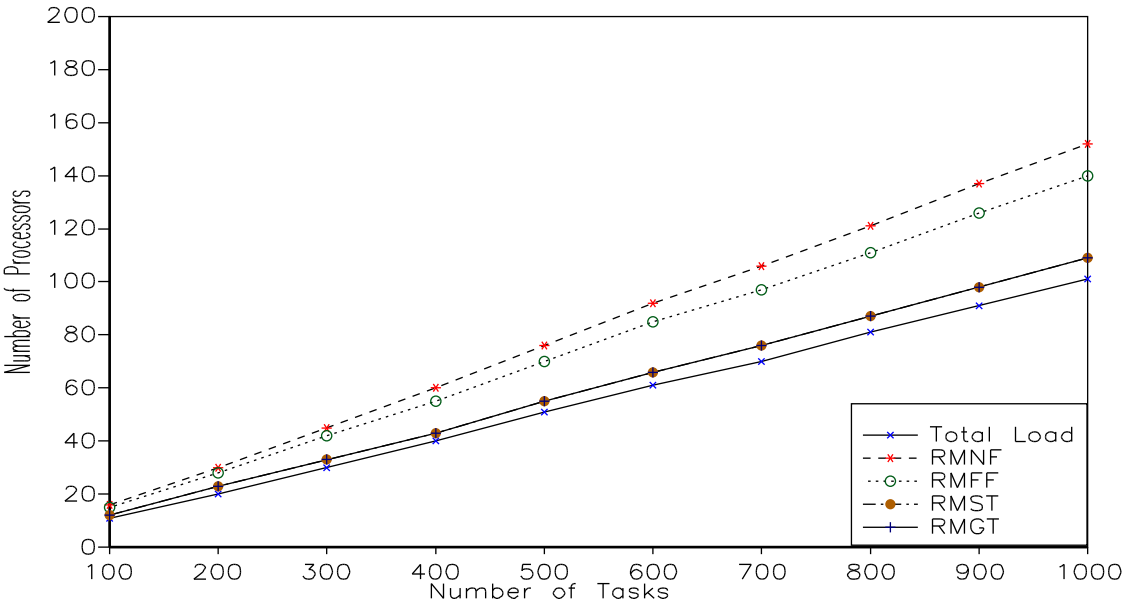

Figure A.1: Task Sets with $\alpha = 0.2$.

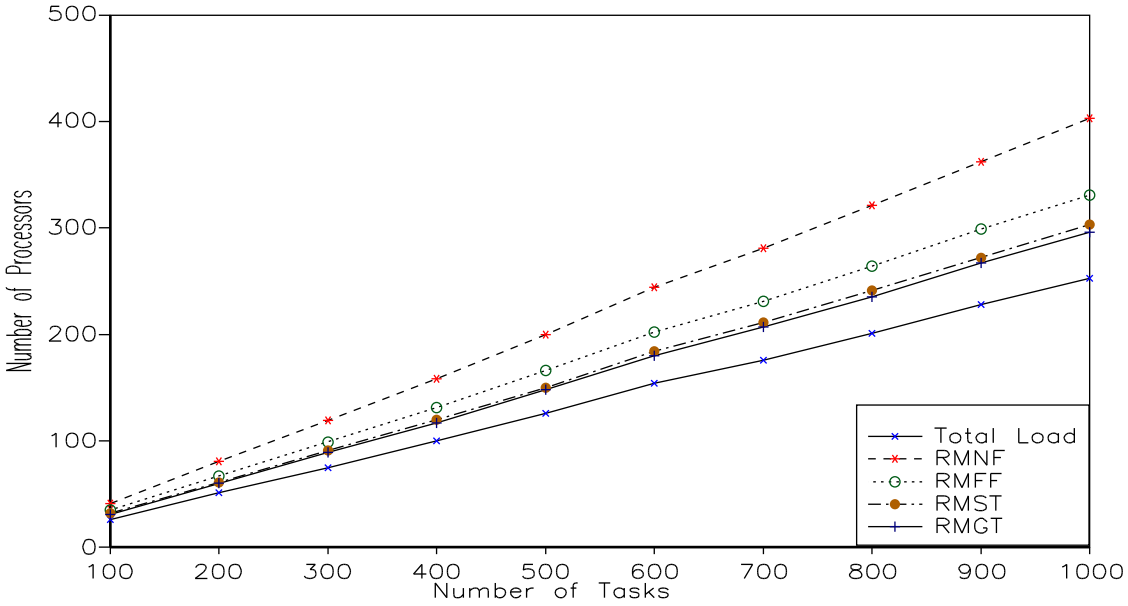

Figure A.2: Task Sets with $\alpha = 0.5$.

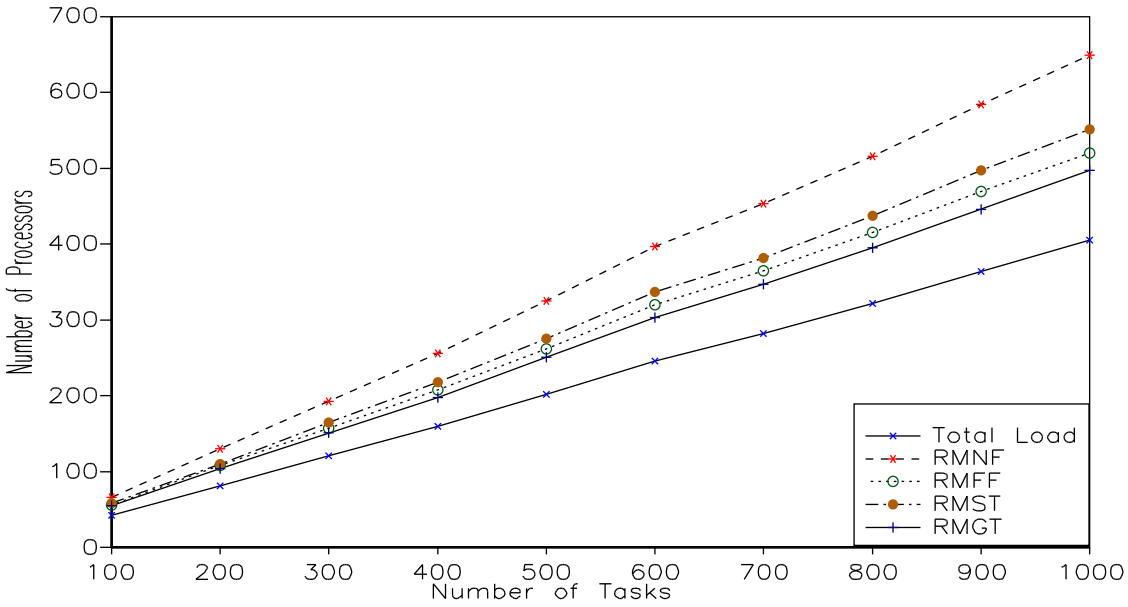

Figure A.3: Task Sets with $\alpha = 0.8$.