# Tight Performance Bounds of Heuristics for a Real-Time Scheduling Problem

Yingfeng Oh and Sang H. Son

Technical Report No. CS-93-24
May 24, 1993

# Tight Performance Bounds of Heuristics for a Real-Time Scheduling Problem

Yingfeng Oh and Sang H. Son
Department of Computer Science
University of Virginia
Charlottesville, VA 22903

## Abstract

The problem of scheduling a set of periodic tasks on a number of processors using a fixed-priority assignment scheme was first studied by Dhall and Liu in their paper entitled “On a real-time scheduling problem”. Two scheduling heuristics — Rate-Monotonic-Next-Fit (RMNF) and Rate-Monotonic-First-Fit (RMFF) were proposed, and their worst-case performance was proven to have an upper bound of 2.67 and 2.2, and a lower bound of 2.4 and 2.0, respectively. In this paper, we first tighten up the worst-case bounds for both RMNF and RMFF, and at the same time, correct some errors existing in the original proof of the upper bound for RMFF. The tight worst-cast bounds of RMNF and RMFF are proven to be 2.67 and 2.33, respectively. Then, in an effort to find a more efficient algorithm, we propose a new scheduling heuristic — Rate-Monotonic-Best-Fit (RMBF), and study its worst-case performance. Surprisingly, RMBF also has a tight worst case bound of 2.33.

## I. Introduction

The problem of preemptively scheduling a set of periodic tasks with hard deadlines equal to the task periods on a single processor was first solved by Liu and Layland [12], and Serlin[17]. In the case of fixed priority assignment, the rate-monotonic algorithm [12] [17] was proven to be optimal. In the case of dynamic priority assignment, the earliest deadline first (*EDF*) algorithm [12] was proven to be optimal. The rate-monotonic algorithm assigns priorities to tasks according to their periods, where the priority of a task is in inverse relationship to its period. The rate-monotonic algorithm has recently gained a lot of recognition since it can be used as a backbone algorithm for designing predictable real-time systems. Many significant results have been obtained

This work was supported in part by ONR, by DOE, and by IBM.

within the framework of rate-monotonic scheduling, for example, the scheduling of tasks which need to be synchronized [20], the scheduling of tasks which are “imprecise” [13] [14] [21], the scheduling of aperiodic and sporadic tasks [9] [22], and the scheduling of support to overcome transient overload [16].

In this paper, we consider the problem of scheduling a set of periodic tasks on a multiprocessor system using a fixed priority assignment scheme. The study of scheduling periodic tasks on a multiprocessor system is justified for various reasons. In some cases, due to heavy computing demands, multiprocessor support can be the best, perhaps the only, means of providing sufficient processing power to meet critical real-time deadlines. In other cases, multiprocessor support for a hard real-time system is a much better choice for reliability than a uniprocessor system, since a multiprocessor system is generally more reliable than a uniprocessor system. However, the problem of scheduling a set of periodic tasks, using either fixed priority or dynamic priority assignment, on a number of processors, is a *NP-hard* problem [11]. For practical purpose, simple scheduling heuristics which can obtain fast results need to be devised.

There are potentially numerous scheduling heuristics to solve this scheduling problem. A particular class of scheduling heuristics, which use the rate-monotonic algorithm to schedule the set of tasks assigned on each individual processor, is especially favored for a number of reasons. First, the rate-monotonic algorithm is optimal for fixed priority assignment of periodic tasks on a processor. The reason the fixed priority assignment is used is for practical purposes, such as ease of implementation and the minimal scheduling overhead involved. Secondly, simple and efficient bin-packing heuristics can be used to assign tasks onto processors so that the least number of processors is used. Finally, since rate-monotonic scheduling is used to schedule tasks on a processor, many extant results concerning rate-monotonic scheduling of real-time tasks on a single processor can be readily adapted to accommodate more practical needs of real-time systems, such as, the scheduling of soft-deadline tasks [10], the scheduling of tasks which need to be synchronized [20], and the mode change protocols [19] [23].

Dhall and Liu [5] were the first to propose heuristic algorithms to solve this problem. They proposed two scheduling algorithms, which were called the Rate-Monotonic-Next-Fit (*RMNF*) and Rate-Monotonic-First-Fit (*RMFF*), and analyzed their performance. The worst-case performance of *RMNF* and *RMFF* was proven to have a upper bound of 2.67 and 2.23, and a lower bound of 2.4 and 2.0, respectively. Unfortunately, the bounds were not tight, with gaps existing between the upper bounds and lower uppers of both algorithms. This posed a nagging problem, since the lower bounds may in fact be the real lower bounds, and the upper bounds may in fact be the real upper bounds, or neither the lower bounds nor the upper bounds have anything to do with the real bounds. This problem may be aggravated in actual implementation when multiple processors are needed to execute a number of periodic tasks in some hard real-time applications. On the one hand, a sufficient number of processors should be provided to execute the tasks so that their hard deadlines are guaranteed even in the worst case. On the other hand, the least number of processors should be used so that processor resources are not wasted. In this paper, we tighten up the worst-case performance bounds for both *RMNF* and *RMFF*. In the process, we found that there were some errors in the original proof of the upper bound for *RMFF*. The errors were corrected, and the tight bound of *RMFF* was shown to be 2.33 rather than 2.23.

In an attempt to find more efficient algorithms, we then propose a new scheduling algorithm — Rate-Monotonic-Best-Fit (*RMBF*), and study its performance. This new algorithm, based on the bin-packing heuristic, Best-Fit, assigns tasks on processors in such a manner as to maximize the utilization of a processor. *RMBF* is intrinsically more complex than *RMFF*, and is expected to have better performance in assigning tasks to processors. However, the worst-case performance of *RMBF* is, to our surprise, no better than that of *RMFF*.

Bin-packing heuristics are chosen because assigning tasks on processors bears many similarity to packing items into bins. Many of the bin-packing heuristics are quite simple, and yet deliver very good performance. The key difference in this case, however, is that bins in bin-packing have unitary size, while the “size” or utilization of a processor in scheduling tasks on a multiprocessor changes dynamically according to some pre-defined functions. This difference makes the analysis of the worst-case performance of the scheduling heuristics considerably more complicated than that of bin-packing heuristics.

Other related work in this area includes the two scheduling heuristics studied by Davari and Dhall [3] [4]. They are the First-Fit-Decreasing-Utilization-Factor (*FFDUF*) and *NEXT-FIT-M* algorithms. *FFDUF* sorts the tasks in non-increasing order of utilization factor and assigns them to processors in that order. *NEXT-FIT-M* classifies tasks into $M$ classes with respect to their utilizations. Processors are also classified into $M$ groups, so that a processor in $k$-group executes tasks in $k$-class exclusively. The worst-case performance of *FFDUF* is tightly bounded by 2, while the performance of *NEXT-FIT-M* is upper bounded by a number $S_M$, which is a decreasing function of the pre-selected number $M$, e.g., $S_M = 2.34$ for $M = 4$, and $S_M = 2.28$ for $M \longrightarrow \infty$.

This paper is organized as follows. In the next section, the scheduling problem is formally defined. The performance of *RMNF* is proven to be tightly bounded by *2.67* in Section III. The *RMFF* algorithm is presented, and its performance analyzed in Section IV, while the performance of *RMBF* is given in Section V. Finally, we conclude in Section VI and indicate the remaining problems.

## II. Problem Definition

In order to present any scheduling results, it is necessary to state the assumptions beforehand. The presentation of these assumptions follows the format used by Liu and Layland [12].

(A) The requests of all tasks for which hard deadlines exist are periodic, with constant interval between request.

(B) Each task must be completed before the next request for it arrives — i.e., all its versions must be completed by the end of each request period.

(C) The tasks are independent in that the requests of a task do not depend on the initiation or the completion of requests for other tasks.

(D) The computation time for each task is constant for that task and does not vary with time. Computation time here refers to the time which is taken by a processor to execute the task without interruption.

(E) Any non-periodic tasks in the system are special, and do not have hard deadlines.

Assumptions (A), (B), and (D) have been argued to have close resemblance to many industrial real-time systems [12], and have thus been used by many in studying and building real-time systems. Though Assumption (C) does exclude the situation where certain tasks have precedence of execution before others, it nevertheless is a good model for many real-time systems. Assumption (E) may appear to be overly restrictive in the first glance. However, with the rapid advance of real-time scheduling techniques, Assumption (E) can be totally omitted. It is put in here so that we can focus ourselves on periodic tasks at the moment. The various ways to efficiently schedule non-periodic, hard deadline tasks along with periodic tasks can be found in real-time scheduling literature [9] [22].

It follows that a task $\tau_i$ is completely characterized by two numbers, the computation time $C_i$ and the period $T_i$. The ratio $C_i / T_i$ is called the utilization factor of the task $\tau_i$. The problem of scheduling a set of periodic tasks on a multiprocessor can be defined as follows: Given a set of $n$ tasks $\Sigma = \{\tau_1, \tau_2, \ldots, \tau_n\}$, where each task $\tau_i$ is characterized by its computation time $C_i$ and its period $T_i$, i.e., $\tau_i = (C_i, T_i)$, what is the minimum number of processors needed to execute the task set such that all $n$ tasks can be guaranteed to meet their deadlines? The preemptive scheduling discipline and the fixed priority assignment scheme are assumed.

To solve this problem, a heuristic approach which consists of two steps is usually adopted: (1) A heuristic algorithm is employed to assign tasks to processors; (2) The rate-monotonic algorithm is used to schedule tasks on each individual processor. The problem of assigning tasks onto a minimal number of processors very closely resembles the bin-packing problem, in which items of variable sizes are packed into as few bins as possible. Therefore, many of the bin-packing heuristics can be used to assign tasks onto processors. However, there is a key difference between bin-packing and the scheduling of periodic tasks on a multiprocessor: the “size” of a bin, which corresponds to the utilization of a processor, is not always unitary, but rather it is a variable whose values are determined by some pre-defined functions. These functions are referred to as *schedulability conditions*.

When a task is assigned to a processor, the scheduler must make sure that the addition of the task to the processor should not jeopardize the schedulability of those tasks that have already been assigned to it. To accomplish this goal, the following schedulability condition can be used.

***Condition WC***: If a set of $m$ tasks is scheduled according to the rate-monotonic scheduling algorithm, then the minimum achievable utilization factor is $m(2^{1/m}-1)$. As $m$ approaches infinity, the minimum utilization factor approaches *ln2*.

This schedulability condition was first given by Liu and Layland [12]. It implies that a task set can be scheduled to meet their deadlines if the total utilization factor of the tasks is less than a threshold number, which is given by $m(2^{1/m}-1)$, where $m$ is the number of tasks to be scheduled. This condition is a worst-case condition, and therefore it is referred to as ***Condition WC*** (*Worst Case*). The function $f(m) = m(2^{1/m}-1)$ is a strictly decreasing function with regards to $m$, the number of tasks on a processor. In studying the performance of *RMNF* and *RMFF*, Dhall and Liu [5] used a different schedulability condition, which is stated as follows:

***Condition IP***: Let $\tau_1, \tau_2, \ldots, \tau_m$ be a set of $m$ tasks with periods $T_1 \le T_2 \le \ldots \le T_m$. Let $u = \sum_{i=1}^{m-1} C_i/T_i \le (m-1)(2^{1/(m-1)}-1)$. If $C_m / T_m \le 2(1 + u/(m-1))^{-(m-1)} - 1$, then the set can be feasibly scheduled by the rate-monotonic scheduling algorithm. As $m$ approaches infinity, the minimum utilization factor of $\tau_m$ approaches $2e^{-u} - 1$.

This schedulability condition requires that the tasks be sorted in the order of non-decreasing period, thus implying that the task set should be known beforehand. Some of the task sets that can not be scheduled by using ***Condition WC*** can be scheduled by using this condition, since this condition takes advantage of the fact that tasks are ordered against non-decreasing periods. This condition is referred to as ***Condition IP*** (*Increasing Period*). The function $f(u, m) = 2(1 + u/(m-1))^{-(m-1)} - 1$ is a strictly decreasing function with regards to both $u$ and $m$. Both ***Condition WC*** and ***Condition IP*** can be easily used to test the schedulability of a task set, since the only parameters involved are the total utilization of tasks and the number of tasks. A sufficient and necessary condition, which takes into account both the computation time and the period of a task when a task is scheduled, was recently given by Lehoczky et al [8]. Because of its complexity, the performance of the scheduling heuristics using this condition is not studied here. In the following, we focus our studies on the scheduling heuristics using ***Condition IP*** as schedulability condition. Note that the scheduling heuristics — *RMNF* and *RMFF* studied by Dhall and Liu used the same schedulability condition.

*Notations*: Let $N_0$ and $N(A)$ be the number of processors used by an optimal algorithm and the number of processors used by a heuristic algorithm A, respectively. Then, the guaranteed performance bound of the algorithm $A$, denoted as $\Re(A)$, is defined as

$$\Re(A) = \lim_{N_0 \to \infty} \frac{N(A)}{N_0}$$

Processors are numbered in the order consistent with that of allocating them. $P$ and $Q$ are used to denote processors. $\tau_{x,\,l}$ denotes the $l$th task that is assigned on the $x$th processor. $u_{x,\,l}$ denotes the utilization of task $\tau_{x,\,l}$. $\tau_i$ is used to denote the $i$th task where there is no confusion. $u_i$ denotes the utilization of the $i$th task on a processor or in a task set. $\tau = (x, y)$ characterizes a task $\tau$, where $x$ and $y$ are the computation time and the period of task $\tau$.

## III. Tight Bound for Rate-Monotonic-Next-Fit

The Rate-Monotonic-Next-Fit algorithm is given as follows:

***Algorithm RMNF***:

1. Tasks are sorted in the non-decreasing order of their periods.
2. Set $i = j = 1$. /* $i$ denotes the $i$th task, $j$ the number of processors allocated */
3. Assign task $\tau_i$ to processor $P_j$ if this task together with the tasks that have been assigned to $P_j$ can be feasibly scheduled on $P_j$ according to ***Condition IP***. If not, assign task $\tau_i$ to $P_{j+1}$ and set $j = j + 1$.
4. If $i < n$, then set $i = i + 1$ and go to step 3 else stop.

When the algorithm finishes, the value in $j$ is the number of processors required to execute a given set of tasks. In order to obtain the tight bound of its worst-case performance, we prove that the upper bound given by Dhall and Liu is indeed the real upper bound by showing that for a given number of processors in the optimal schedule, a task set which can achieve the worst-case upper bound under ***Algorithm RMNF*** can always be constructed. The upper bound was stated in Theorem 3.1, the proof of which can be found in [12]. The low bound, as given in Theorem 3.2, requires surprisingly a much involved proof.

***Theorem 3.1:*** For all sets of tasks, $\lim_{N_0 \to \infty} N/N_0 \le 2.67$, where $N_0$ is the minimum number of processors required to feasibly schedule the same set of tasks, and $N$ is the number of processors obtained by ***Algorithm RMNF***.

***Theorem 3.2:*** Let $N$ be the number of processors required to feasibly schedule a set of tasks by ***Algorithm RMNF***, and $N_0$ the minimum number of processors required to feasibly schedule the same set of tasks. Then $\lim_{N_0 \to \infty} N/N_0 \ge 2.67$. Together with ***Theorem 3.1***, it is concluded that $\Re(RMNF)^{0} = 2.67$.

***Proof:*** In order to find the worst-case situations, where the biggest ratio between $N$ and $N_0$ is achieved, it is necessary to find those sets of tasks, which, when scheduled by ***Algorithm RMNF***, use as many processors as possible. In other words, for a given set of tasks, where the total utilization is fixed, the worst case is achieved by appropriately allocating the utilization for each task and ordering the tasks in a certain way such that the number of processors required to execute the task set is maximized according to the ***RMNF Algorithm***.

The function $f(u, m) = 2(1 + u/(m-1))^{-(m-1)} - 1$ is a strictly decreasing function with regards to $m$, and it approaches the minimum utilization factor given by $2e^{-u} - 1$ when $m$ approaches infinity. In fact, for a sufficiently large number $m$, $2(1 + u / (m-1))^{-(m-1)} - 1$ approaches $2e^{-u} - 1$ very quickly. Therefore, we claim that the following set of $N^*(m + 1)$ tasks requires $N$ processors for sufficiently large $m$ and small $\varepsilon$, when scheduled by ***Algorithm RMNF***:

$$(\alpha, 1), \underbrace{(\varepsilon, 1), \ldots\ldots, (\varepsilon, 1)}_{m}, \ldots\ldots, (\alpha, 1), \underbrace{(\varepsilon, 1), \ldots\ldots, (\varepsilon, 1)}_{m}$$

where the value of $\alpha$ is obtained by solving the equation $\alpha = 2e^{-\alpha} - 1$. $\alpha \approx 0.3745$. That $N$ processors are used by ***Algorithm RMNF*** to schedule the $N^*(m + 1)$ tasks is because $\alpha > 2e^{-(\alpha+m\varepsilon)} - 1$. The total utilization of this set of tasks is therefore equal to $N\alpha + Nm\varepsilon$. If this task set can be perfectly scheduled on $N_0 = N\alpha + Nm\varepsilon$ in the optimal schedule, then $\Re = \frac{N}{N_0} = N / (N\alpha + Nm\varepsilon) \approx 1 / \alpha \approx 2.67$, for very small $\varepsilon$ such that $m\varepsilon$ is small. Unfortunately, since $\alpha \approx 0.3745$, the above task set can not be perfectly scheduled in the optimal schedule using only $N\alpha + Nm\varepsilon$ processors, unless the execution of a task can be interrupted (not because of the rate-monotonic property), and its execution be resumed on another processor immediately. This later requirement is often referred to as processor migration. This implies that rate-monotonic algorithm is not honored in the optimal schedule.

Though the above example does not suit our purpose, there are a number of things that we can adopt from the above example in finding the worst-case examples. First, the last tasks assigned to each processor in the completed ***RMNF*** schedule is a very large number $m$ of tasks each with a very small utilization $\varepsilon$ such that $m\varepsilon$ is small. Then the equation $f(u) = 2e^{-u} - 1$ is used as the schedulability test condition. Second, on each processor in the completed ***RMNF*** schedule, it is always assigned, as the first task, a task with a large utilization (compared to $\varepsilon$), followed by, with few exceptions, $m$ tasks each with a very small utilization $\varepsilon$ subsequently. From now on, we only concern ourselves with the utilization of the first task on each of processors in the completed ***RMNF*** schedule. The following set of the tasks (tasks with $\varepsilon$ utilization excluded) gives the worst-case performance of ***Algorithm RMNF***:

(0.402764, 1), (0.336940, 1), (0.427903, 1), (0.303749, 1),

(0.476093, 1), (0.242412, 1), (0.569466, 1), (0.131655, 1), (0.223080, 1)

(0.402764, 1), (0.336940, 1), (0.427903, 1), (0.303749, 1),

(0.476093, 1), (0.242412, 1), (0.569466, 1), (0.131655, 1).

The utilization of $u_{i+1}$ is given by $2e^{-u_i} - 1$ for $1 \le i \le 7$ and $9 \le i \le 16$. According to the reasons given above, the first 8 tasks (tasks with $\varepsilon$ utilization excluded) occupy 8 processors in the completed ***RMNF*** schedule, the 9th task is scheduled on the 8th processor, and the rest of the

8 tasks are scheduled on 8 processors, for the same reasons. Therefore the total number of processors used in the completed ***RMNF*** schedule is 16. Excluding the 9th task, these 16 tasks can be optimally scheduled on 6 processors, as shown below:

Processor 1: tasks 1 and 5. Processor 2: tasks 2 and 6. Processor 3: tasks 3 and 4. Processor 4: tasks 7, 8, and 16. Processor 5: tasks 10, 11, and 12. Processor 6: tasks 13, 14, 15, and 17. The total utilizations of these processors are 0.997369, 0.997369, 0.952186, 0.937183, 0.977629, and 0.920228, respectively. Obviously task 9 can not be scheduled into any of these 6 processors, though the total available processor utilization on the 6 processors is larger than the utilization of the 9th task. If the 9th task is replaced by a number of tasks each with a small utilization, yet their total utilizations add up to 0.223080, then the number of processors required to execute this new set of tasks is still 6 in the optimal schedule, since those newly replacing tasks can be now scheduled on the 6 processors. The replacement of the 9th task does not change the number of processors used in the ***RMNF*** schedule either. Therefore, $\Re = \frac{N}{N_0} = 16 / 6 \approx 2.67$.

Yet, in order to prove the theorem, we need to show that for any given number $N_0$, a task set can be constructed such that the ratio 2.67 is achieved. Even though we only give one task set above as the example where this worst-case ratio is indeed achieved, we claim that the 2.67 ratio is indeed achievable for different numbers of $N_0$. For any given number $N_0$, the task set that can achieve the worst-case ratio can be constructed. However, the construction has to be done in a case by case manner, similar to the above example. The claim lies in the fact that, if the utilization of a task (tasks with $\varepsilon$ utilization excluded) is given by $u_{i+1} = 2e^{-u_i} - 1$ for $1 \le i \le N\text{-}1$, and $u_1 > \alpha$, then the total utilization of the $N$ tasks is given by $N\alpha + Nm\varepsilon$. The ratio is then given by $\Re = \frac{N}{N_0} < N / (N\alpha + Nm\varepsilon) \approx 2.67$. The key, of course, is to find many sequences of $u_i$s such that they can be perfectly scheduled in the optimal scheduled without requiring process migration. Note that with $u_1 > \alpha \approx 0.3745$, and $u_{i+1} = 2e^{-u_i} - 1$ for $1 \le i \le N - 1$, $u_{2i-1} + u_{2i} < 2\alpha$ for $1 \le i \le \lfloor N/2 \rfloor$, as shown in Figure 1. $\sum_{i=1}^{N} u_i \le N\alpha$ if $N$ is even. $\sum_{i=1}^{N} u_i \ge N\alpha$ if $N$ is odd, since $u_{2i} + u_{2i+1} > 2\alpha$ for $1 \le i \le \lfloor (N-1)/2 \rfloor$.

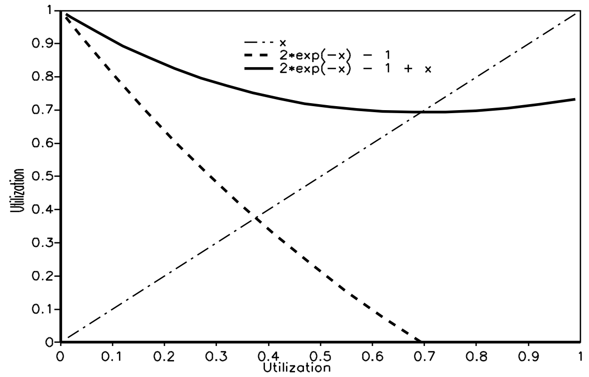


**Figure 1**: Properties of ***Condition IP***

From ***Theorem 3.1***, it is concluded that $\Re = \lim_{m \to \infty} \frac{N}{N_0} = 2.67$. Q.E.D.

## IV. Tight Bound for Rate-Monotonic-First-Fit

In assigning tasks to processors, ***Algorithm RMNF*** only checks the current processor to see whether a task together with those tasks that have already been assigned to that processor can be feasibly scheduled or not. If not, the task has to be scheduled on an idle processor, even though the task may be scheduled on those processors used earlier. To overcome this waste of processor utilization, the ***RMFF Algorithm*** always starts to check the schedulability of a task on processors with lower indexes, i.e., those processors on which some tasks have been assigned. This algorithm is given as follows:

***Algorithm RMFF***: Let the processors be indexed as $P_1, P_2, \ldots$, with each initially in the idle state, i.e., with zero utilization. The tasks $\tau_1, \tau_2, \ldots, \tau_n$, which are ordered according to non-decreasing periods, will be scheduled in that order. To schedule $\tau_i$, find the least $j$ such that task $\tau_i$, together with all the tasks that have been assigned to processor $P_j$ can be feasibly scheduled according to ***Condition IP*** for a single processor, and assign task $\tau_i$ to $P_j$.

***Algorithm RMFF*** can be described in a more algorithmic format as follows:

***Algorithm RMFF*** (Input: task set $\sum$; Output: $m$)

1. Tasks are sorted in the non-decreasing order of their periods.
2. Set $i = 1$ and $m = 1$. /* $i$ denotes the $i$th task, $m$ the number of processors allocated*/
3. (a) Set $j = 1$. /* $j$ denotes the $j$th processor */

   (b) If $u_i \leq 2\,(1 + U_j/k_j)^{-k_j} - 1$, assign task $\tau_i$ to $P_j$, i.e., increment $k_j = k_j + 1$ and $U_j = U_j + u_i$, and set $m = j$ if $j < m$, where $k_j$ and $U_j$ denote the number of tasks already assigned to processor $P_j$ and the total utilization of the $k_j$ tasks, respectively, and $u_i$ denotes the utilization of task $\tau_i$. Otherwise, increment $j = j + 1$ and go to step 3(b).
4. If $i > n$, i.e., all tasks have been assigned, then return $m$. Otherwise increment $i = i + 1$ and go to step 3(a).

When the algorithm terminates, $m$ is the number of processors required for scheduling the given set of tasks according to the ***RMFF Algorithm***. Since an idle processor will not be used until all the processors with some utilizations can not execute an incoming task, it is therefore expected that ***Algorithm RMFF*** would have better performance than that of ***Algorithm RMNF***, which was shown to be the case, to some extent, by Dhall and Liu [5]. The following results were obtained in [5].

***Lemma 1***: If $m$ tasks can not be feasibly scheduled on $m-1$ processors according to the ***RMFF Algorithm***, then the utilization factor of the set of tasks is greater than $m/(1+2^{1/3})$.

***Lemma 2***: If tasks are assigned to the processors according to the ***RMFF Algorithm***, among all processors to each of which two tasks are assigned, there is at most one processor for which the utilization factor of the set of the two tasks is less than 1/2.

***Theorem 1***: Let $N$ be the number of processors required to feasibly schedule a set of tasks by the ***RMFF Algorithm***, and $N_0$ the minimum number of processors required to feasibly schedule the same set of tasks. Then as $N_0$ approaches infinity, $2 \le N/N_0 \le 4 \times 2^{1/3} / (1 + 2^{1/3})$ ($\approx 2.23$).

Unfortunately, Lemma 1 is incorrect, as shown by the following counter example. Lemma 2 gives a weak result for ***RMFF Algorithm***. These two errors led the authors to arrive at the wrong upper bound. In the following, we first show the incorrectness of Lemma 1, and present its correct version. We then give a strong version of Lemma 2. A new upper bound is proven finally.

***Example***: Consider the case where $m = 2$ and the two tasks are given as follows:

$\tau_1 = (2^{1/2} - 1, 1)$

$\tau_2 = (2 - 2^{1/2} + \varepsilon, 2^{1/2})$, where $\varepsilon$ is a small number and $\varepsilon > 0$.

According to the ***RMFF Algorithm***, $\tau_1$ is first assigned to a processor. Since $u_1 = 2^{1/2} - 1$ and $2(1 + u_1)^{-1} - 1 = 2^{1/2} - 1 < 2^{1/2} - 1 + \varepsilon / 2^{1/2} = u_2$, $\tau_2$ can not be scheduled together with task $\tau_1$ on one processor, according to ***Condition IP***. Since $\tau_1$ and $\tau_2$ can not be scheduled on one processor, $u_1 + u_2$ must be greater than $2/(1 + 2^{1/3}) \cong 0.88$ according to Lemma 1. But $u_1 + u_2 = 2(2^{1/2} - 1) + \varepsilon / 2^{1/2} = 0.8284 + \varepsilon / 2^{1/2}$, which is less than 0.88 for small $\varepsilon$.

When $m > 2$, similar examples can be constructed to show the incorrectness of Lemma 1. Henceforth, a new version of Lemma 1 is given as follows:

***Lemma 4.1***: If $m$ tasks can not be feasibly scheduled on $m - 1$ processors according to the ***RMFF Algorithm***, then the utilization factor of the set of tasks is greater than $m / (1 + 2^{1/2}) = m(2^{1/2} - 1)$.

***Proof***: The proof is by induction.

(1) $m = 2$. Suppose $u_1$ and $u_2$ are the utilizations of two tasks which can not be scheduled on a processor according to ***Condition IP***, i.e., $u_2 > 2(1 + u_1)^{-1} - 1$. $u_1 + u_2 = u_1 + 2(1 + u_1)^{-1} - 1$. To find the minimum of $f(u_1) = u_1 + 2(1 + u_1)^{-1} - 1$, we take the derivative of function $f(u_1)$, and solve for $u_1$. The minimum of $f(u_1)$ is achieved when $u_1 = (2^{1/2} - 1)$. Therefore $u_1 + u_2 > 2(2^{1/2} - 1)$.

(2) Suppose the Lemma is true for $m = k$, i.e.,

$$\sum_{i=1}^{k} u_i > k(2^{1/2} - 1) \qquad \textbf{\textit{(E.Q.1)}}$$

where $u_i$ is the utilization of task $i$.

When $m = k + 1$, the $(k + 1)$th task can not be scheduled on any of the $k$ processors, i.e., $u_i + u_{k+1} > 2(2^{1/2} - 1)$, where $1 \le i \le k$. Summing up the $k$ equations yields

$$\sum_{i=1}^{k} u_i + k u_{k+1} > 2k(2^{1/2} - 1) \qquad \textbf{\textit{(E.Q.2)}}$$

Multiplying $k$ *-1* on both sides of equation (E.Q.1) yields

$$(k\text{ -}1) \sum_{i=1}^{k} u_i > (k - 1)\ k\,(2^{1/2} - 1) \qquad \textbf{(E.Q.3)}$$

Adding up equations (E.Q.2) and (E.Q.3) and dividing the new equation on both sides by $k$ yields $\sum_{i=1}^{k+1} u_i > (k+1)\ (2^{1/2} - 1)$. Therefore Lemma 4.1 is proven. Q.E.D.

A strong version of the original lemma — Lemma 2 by Dhall and Liu is given as follows:

***Lemma 4.2***: If tasks are assigned to the processors according to the ***RMFF Algorithm***, among all processors to each of which two tasks are assigned, there is at most one processor for which the total utilization factor of the two tasks is less than or equal to $2(2^{1/3}-1) \approx$ 0.52.

***Proof:*** This lemma is proven by contradiction. Suppose that the contrary is true. Let $\tau_{j,1}$ and $\tau_{j,2}$ be the two tasks assigned to processor $P_j$, and $\tau_{k,1}$ and $\tau_{k,2}$ be the two tasks assigned to processor $P_k$ with $j < k$, such that

$$u_{j,1} + u_{j,2} \leq 2(2^{1/3}-1)$$

and

$$u_{k,1} + u_{k,2} \leq 2(2^{1/3}-1), \qquad \textbf{(E.Q.4)}$$

where $u_{x,l}$ is the utilization of task $\tau_{x,l}$.

There are three cases to consider. Note that the testing condition used is ***Condition IP***, i.e., if a task's utilization $C/T \leq 2\,(1 + u/(m-1))^{-(m-1)}$ - 1, then this task together with the $m$ - *1* tasks which have already been assigned to a processor can be feasibly scheduled by the rate-monotonic scheduling algorithm, where $u = \sum_{i=1}^{m-1} C_i/T_i \leq (m-1)\ (2^{1/(m-1)} - 1)$. The function $f(u, m) = 2\,(1 + u/(m-1))^{-(m-1)}$ - 1 is a strictly decreasing function with regards to $u$ and $m$.

Case 1: Tasks $\tau_{k,1}$ and $\tau_{k,2}$ were assigned to processor $P_k$ after task $\tau_{j,2}$ had been assigned to processor $P_j$. According to ***RMFF***, we must have

$$u_{k,1} > 2(1 + (u_{j,1} + u_{j,2})/2)^{-2} - 1$$

and

$$u_{k,2} > 2(1 + (u_{j,1} + u_{j,2})/2)^{-2} - 1$$

Summing up these two inequalities, we have

$$u_{k,1} + u_{k,2} > 4(1 + (u_{j,1} + u_{j,2})/2)^{-2} - 2 > 4(1 + 2^{1/3}-1)^{-2} - 2 = 2(2^{1/3}-1)$$

which is a contradiction to (E.Q.4).

Case 2: Tasks $\tau_{k,1}$ and $\tau_{k,2}$ were assigned to processor $P_k$ after task $\tau_{j,1}$ had been assigned to processor $P_j$, but before task $\tau_{j,2}$. According to ***RMFF***, we must have

$$u_{k,1} > 2(1 + u_{j,1})^{-1} - 1$$

and

$$u_{k,2} > 2(1 + u_{j,1})^{-1} - 1$$

Summing up these two inequalities yields

$$u_{k,1} + u_{k,2} > 4(1 + u_{j,1})^{-1} - 2 > 4(1 + 2(2^{1/3}-1))^{-1} - 2) > 2(2^{1/3}-1)$$

since $u_{j,1} \leq 2(2^{1/3}-1)$ and $2(1 + 2(2^{1/3}-1))^{-1} > 2^{1/3}$. However, this is again a contradiction to (E.Q.4).

Case 3: Task $\tau_{k,1}$ was assigned to processor $P_k$ after task $\tau_{j,1}$ had been assigned to processor $P_j$, and task $\tau_{k,2}$ was assigned to $P_k$ after task $\tau_{j,2}$ had been assigned to $P_j$. According to ***RMFF***, we must have

$$u_{k,1} > 2(1 + u_{j,1})^{-1} - 1$$

and

$$u_{k,2} > 2(1 + (u_{j,1} + u_{j,2}) / 2)^{-2} - 1 > 2(1 + 2^{1/3}-1)^{-2} - 1 = (2^{1/3}-1)$$

Summing up these two inequalities yields

$$u_{k,1} + u_{k,2} > 2(1 + u_{j,1})^{-1} - 1 + (2^{1/3}-1) > 2(1 + 2(2^{1/3}-1))^{-1} - 1 + (2^{1/3}-1) > 2^{1/3}- - 1 + (2^{1/3}-1) = 2(2^{1/3}-1)$$

which is again a contradiction to (E.Q.4). Q.E.D.

Actually, a more generalized result is obtained for the case where the number of tasks assigned to a processor is arbitrary. The proof of the following lemma is given in the appendix.

***Lemma 4.3***: If tasks are assigned to the processors according to the ***RMFF Algorithm***, among all processors to each of which $n \geq 1$ tasks are assigned, there is at most one processor for which the utilization factor of the $n$ tasks is less than or equal to $n(2^{1/(n+1)}-1)$. $\lim_{n \to \infty} n(2^{1/(n+1)} - 1) = ln 2$

***Theorem 4.1***: Let $N$ be the number of processors required to feasibly schedule a set of tasks by the ***Algorithm RMFF***, and $N_0$ the minimum number of processors required to feasibly schedule the same set of tasks. Then $\lim_{N_0 \to \infty} N/N_0 \leq 2 + (3 - 2^{3/2}) / (2(2^{1/3} - 1)) \approx 2.33$.

In order to prove the above bound, we define a function that maps the utilizations of tasks into the real interval [0, 1] as follows:

$$f(u) = \begin{cases} u/(2(2^{1/3}-1)) & 0 \leq u < 2(2^{1/3}-1) \\ 1 & 2(2^{1/3}-1) \leq u \leq 1 \end{cases}$$

or

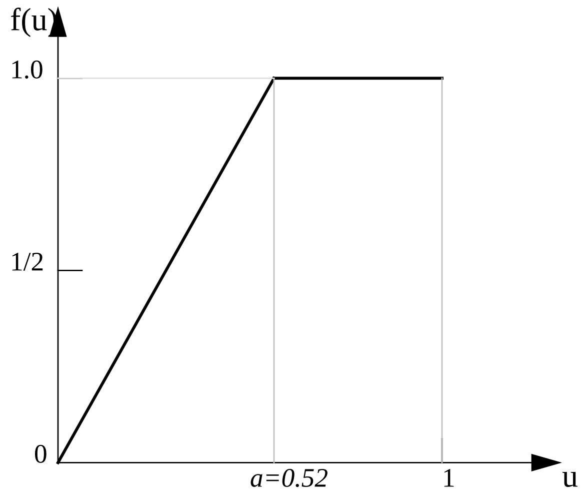


**Figure 2**: Mapping Function for ***RMFF*** and ***RMBF***

$$f(u) = \begin{cases} u/a & 0 \le u < a \\ 1 & a \le u \le 1 \end{cases}$$, where $a = 2(2^{1/3} - 1)$.

Let $\tau_{j,1}, \tau_{j,2}, \ldots, \tau_{j,k_j}$ be $k_j$ tasks assigned to processor $P_j$, and let $\sum_{i=1}^{k_j} u_{j,i} = U_j$. The deficiency $\delta_j$ of processor $P_j$ is defined as

$$\delta_j = \begin{cases} 0 & U_j \ge k_j (2^{1/k_j} - 1) \\ 2(1 + U_j/k_j)^{-k_j} - 1 & Otherwise \end{cases}$$

The coarseness $\alpha_j$ of processor $P_j$ is defined as

$$\alpha_j = \begin{cases} 0 & j = 1 \\ max_{1 \le i \le j-1} \delta_i & j > 1 \end{cases}$$

***Lemma 4.4***: For ***Algorithm RMFF***, the following properties hold:

(1) No task is assigned to an idle processor unless it can not be assigned in any non-idle processor.

(2) If a processor *P* has a coarseness of $\alpha$, then the utilization of each task that was assigned to *P* exceeds $\alpha$.

***Proof***: For ***Algorithm RMFF***, properties (1) and (2) hold according to its definition. Q.E.D.

***Lemma 4.5***: If a processor is assigned a number of tasks $\tau_1, \tau_2, \ldots, \tau_m$, with utilizations $u_1, u_2, \ldots, u_m$, then $\sum_{i=1}^{m} f(u_i) \le 1/a$, where $a = 2(2^{1/3} - 1)$.

***Proof***: Without lose of generality, it is assumed that $u_1 \ge u_2 \ge \ldots \ge u_m$. If $u_1 \ge a$, then $u_2 < a$. $\sum_{i=1}^{m} f(u_i) = f(u_1) + \sum_{i=2}^{m} f(u_i) = 1 + (\sum_{i=2}^{m} u_i) / a \le 1 + (1 - a) / a = 1 / a$. Otherwise ($u_1 < a$), then $\sum_{i=1}^{m} f(u_i) = \sum_{i=1}^{m} u_i / a \le 1 / a$. Q.E.D.

***Lemma 4.6***: Suppose tasks are assigned to processors according to ***RMFF Algorithm.*** If a processor with coarseness $\alpha \ge a / 3$ is assigned $m \ge 3$ tasks, then $\sum_{i=1}^{m} f(u_i) \ge 1$, where $u_1, u_2, \ldots, u_m$ are utilizations of the *m* tasks $\tau_1, \tau_2, \ldots, \tau_m$ that are assigned to the processor.

***Proof***: According to Lemma 4.4, $u_i > \alpha \ge a / 3$ for $1 \le i \le m$. If one of the tasks has a utilization greater than *a*, then $\sum_{i=1}^{m} f(u_i) \ge 1$. Otherwise, $\sum_{i=1}^{m} f(u_i) = \sum_{i=1}^{m} u_i / a \ge m(a/3) / a \ge 1$, since $m \ge 3$. Q.E.D.

***Lemma 4.7***: Suppose tasks are assigned to processors according to ***RMFF Algorithm.*** If a processor with coarseness $\alpha < a / 3$ is assigned $m \ge 3$ tasks $\tau_1, \tau_2, \ldots, \tau_m$ with utilizations $u_1, u_2, \ldots, u_m$, and $\sum_{i=1}^{m} u_i \ge ln2 - \alpha$, then $\sum_{i=1}^{m} f(u_i) \ge 1$.

***Proof***: If one of the tasks $\tau_1, \tau_2, \ldots, \tau_m$ has a utilization greater than *a*, then $\sum_{i=1}^{m} f(u_i) \ge 1$. Otherwise, $\sum_{i=1}^{m} f(u_i) = \sum_{i=1}^{m} u_i / a \ge (ln2 - \alpha) / a \ge (ln2 - a/3) / a \ge 1$. Q.E.D.

***Lemma 4.8***: Suppose tasks are assigned to processors according to ***RMFF Algorithm.*** If a processor with coarseness $\alpha$ is assigned $m \ge 1$ tasks $\tau_1, \tau_2, \ldots, \tau_m$ with utilizations $u_1, u_2, \ldots, u_m$, and $\sum_{i=1}^{m} f(u_i) = 1 - \beta$ where $\beta > 0$, then

(1) $m = 1$ and $u_1 < a$ or

(2) $m = 2$ and $u_1 + u_2 < a$ or

(3) $m \geq 3$ and $\sum_{i=1}^{m} u_i \leq ln2 - \alpha - a\beta$.

***Proof***: (1) If $m = 1$ and $u_1 \geq a$, then $\sum_{i=1}^{m} f(u_i) \geq 1$, which is a contradiction.

(2) If $m = 2$ and $u_1 + u_2 \geq a$, then $\sum_{i=1}^{m} f(u_i) \geq 1$, which is again a contradiction.

(3) If properties (1) and (2) do not hold, then $m \geq 3$. Since $\sum_{i=1}^{m} f(u_i) < 1$, $\alpha$ must be less than $a / 3$ and $\sum_{i=1}^{m} u_i < ln2 - \alpha$ according to Lemma 4.6 and Lemma 4.7. Let $\sum_{i=1}^{m} u_i = ln2 - \alpha - \gamma$, where $\gamma > 0$. To find out the relationship between $\gamma$ and $\beta$, let us replace the first three tasks $\tau_1$, $\tau_2$, and $\tau_3$ by three new tasks with utilizations $\upsilon_1$, $\upsilon_2$, and $\upsilon_3$, such that $\upsilon_1 + \upsilon_2 + \upsilon_3 = u_1 + u_2 + u_3 + \gamma$, $\upsilon_1 \geq u_1$, $\upsilon_2 \geq u_2$, $\upsilon_3 \geq u_3$, and $\upsilon_1 < a$, $\upsilon_2 < a$, $\upsilon_3 < a$. According to Lemma 4.7, $f(\upsilon_1) + f(\upsilon_2) + f(\upsilon_3) + \sum_{i=4}^{m} f(u_i) \geq 1$. Since $f(\upsilon_1) + f(\upsilon_2) + f(\upsilon_3) = f(u_1) + f(u_2) + f(u_3) + f(\gamma) = f(u_1) + f(u_2) + f(u_3) + \gamma / a$, $\gamma / a + 1 - \beta \geq 1$. $\gamma \geq a\beta$. Therefore, $\sum_{i=1}^{m} u_i \leq ln2 - \alpha - a\beta$. Q.E.D.

Proof of Theorem 4.1: Let $\Sigma = \{\tau_1, \tau_2, \ldots, \tau_m\}$ be a set of $m$ tasks, with their utilizations $u_1, u_2, \ldots, u_m$ respectively, and $\varpi = \sum_{i=1}^{m} f(u_i)$. By Lemma 4.5, $\varpi \leq N_0 / a$, where $a = 2(2^{1/3} - 1)$.

Suppose that among the $N$ number of processors used by ***RMFF Algorithm*** to schedule a given set $\Sigma$ of tasks, $L$ of them has $\sum_j f(u_j) = 1 - \beta_i$ with $\beta_i > 0$, where $j$ ranges over all tasks in processor $i$ among the $L$ processors. Let us divide these processors into three different classes:

(1) Processors that only one task is assigned. Let $n_1$ denote the number of processors in this class.

(2) Processors that two tasks are assigned. Let $n_2$ denote the number of processors in this class. According to Lemma 4.2, there is at most one processor whose utilization in the ***RMFF*** schedule is less than or equal to $a = 2(2^{1/3} - 1)$. Therefore $n_2 = 0$ or 1.

(3) Processors that at least three tasks are assigned. Let $n_3$ denote the number of processors in this class.

Obviously, $L = n_1 + n_2 + n_3$. For each of the rest $N - L$ processors, $\sum_j f(u_j) \geq 1$, where $j$ ranges over all tasks in a processor.

For the processors in class (1), $\sum_{i=1}^{n_1} u_i > n_1 (2^{1/2} - 1)$ according to Lemma 4.1. Since $\sum_{i=1}^{n_1} f(u_i) < 1$, $u_i < a$, and therefore $\sum_{i=1}^{n_1} f(u_i) > n_1 (2^{1/2} - 1) / a$. Moreover, according to Lemma 4.9, there is at most one task whose utilization is less than or equal to $(2^{1/2} - 1)$. In the optimal assignment of these tasks, the optimal number $N_0$ of processors used can not be less than $n_1 / 2$, i.e., $N_0 \geq n_1 / 2$, since possibly with one exception, any three tasks among these tasks can not be scheduled on one processor.

For the processors in class (3), let $Q_1, Q_2, \ldots\ldots, Q_{n_3}$ denote the $n_3$ processors in this class, and $\alpha_i$ be the coarseness of processor $Q_i$, and $\sum_{l=1}^{k_i} f(u_l) = 1 - \beta_i$ with $\beta_i > 0$, for $1 \le i \le n_3$. For processor $i$, $U_i \le ln2 - \alpha_i - a\beta_i$ according to Lemma 4.8.

According to the definition of coarseness, $\alpha_{i+1} \ge \delta_i \ge ln2 - U_i$, since $\delta_i = 2\,(1 + U_i/k_i)^{-k_i} - 1 > 2e^{-u_i} - 1 > ln2 - U_i$. Therefore $\alpha_{i+1} \ge \alpha_i + a\beta_i$, for $1 \le i < n_3$. Summing up these $(n_3 - 1)$ equations yields

$$a\sum_{i=1}^{n_3-1}\beta_i \le \alpha_{n_3} - \alpha_1 < a/3, \text{ i.e., } \sum_{i=1}^{n_3-1}\beta_i < 1/3.$$

$$\sum_{i=1}^{n_3}\sum_{l=1}^{k_i} f(u_l) \ge n_3 - 1 - \sum_{i=1}^{n_3-1}\beta_i > n_3 - 4/3.$$

Now we are ready to find out the relationship between $N$ and $N_0$.

$$\begin{aligned}\varpi = \sum_{i=1}^{m} f(u_i) &\ge (N - L) + n_1\,(2^{1/2} - 1)/a + n_3 - 4/3\\ &= N - n_1 - n_2 - n_3 + n_1\,(2^{1/2} - 1)/a + n_3 - 4/3\\ &= N - n_1(1 - (2^{1/2} - 1)/a) - n_2 - 4/3\\ &\ge N - 2N_0(1 - (2^{1/2} - 1)/a) - n_2 - 4/3, \text{ where } a = 2\,(2^{1/3} - 1).\end{aligned}$$

Since $\varpi \le N_0/a$ by Lemma 4.5,

$$N_0/a \ge N - 2N_0(1 - (2^{1/2} - 1)/a) - n_2 - 4/3 \ge N - 2N_0(1 - (2^{1/2} - 1)/a) - 7/3.$$

Therefore, $N/N_0 \le (2a + 1 - 2(2^{1/2} - 1))/a + 7/(3N_0)$.

$$\lim_{N_0 \to \infty} N/N_0 \le (2a + 1 - 2(2^{1/2} - 1))/a \approx 2.33. \text{ Q.E.D.}$$

***Theorem 4.2***: Let $N$ be the number of processors required to feasibly schedule a set of tasks by ***RMFF Algorithm***, and $N_0$ the minimum number of processors required to feasibly schedule the same set of tasks. Then $\lim_{N_0 \to \infty} N/N_0 \ge 2.3$.

***Proof:*** In order to find the bound $\mathfrak{R} = \lim_{N_0 \to \infty} N/N_0$, we proceed by finding the maximum number of processors needed to schedule a certain set of tasks using ***RMFF Algorithm***, given that the optimal number of processors required to schedule the same set of tasks is known. In the process, the desired set of tasks is constructed. Note that this process is exactly opposite to how a set of tasks is scheduled.

Let $N_0 = m$, where $m$ is a natural number. A set of tasks, which uses exactly $N_0$ number of processors in the optimal schedule, is to be specified in the following. Without generality, all tasks are assumed to have a period of 1. This set of tasks consists of a theoretically infinite regions, given that $N_0$ is sufficiently large. The regions of tasks are given as follows. Note that the regions specified first are scheduled last in the ***RMFF Algorithm***, in other words, they appear last in the task set.

Region 1: There are $2N_0$ number of tasks each with a utilization of $u_1 = (2^{1/2} - 1) + \varepsilon$, where

$\varepsilon$ is a arbitrary small number. These $2N_0$ tasks will utilize $2N_0$ number of processors in the ***RMFF*** schedule, while requires only $N_0$ number of processors in the processors in the optimal schedule. If $N_0 \le 2$, then we have found $\Re = 2$.

Region 2: If $3 \le N_0 \le 5$, there are $N_0$ tasks, each of which has a utilization of $u_2 = (2^{1/5} - 1)$. These $N_0$ tasks utilize one processors in the ***RMFF*** schedule, while requires no extra processor in the optimal schedule, only to fill part of the utilization left by tasks in region 1, i.e., $(2^{1/5} - 1) < 1 - 2*((2^{1/2} - 1) + \varepsilon)$. Note that tasks in region 1 can not be scheduled on this processor, since $u_1 > 2(1 + 3u_2 / 3)^{-3} - 1$. $N = 2N_0 + 1$. The bound is given by $\Re = 2N_0 / N_0 + 1 / N_0$.

Region 3: If $6 \le N_0 \le 9$, the tasks in regions 1 and 2 are included. Furthermore, there are three more tasks each having a utilization of $(2^{1/5} - 1)$ and six tasks each with a utilization of $u_3 = 1 - 2*((2^{1/2} - 1) + \varepsilon) - (2^{1/5} - 1) - \varepsilon$. These nine tasks use one processor in the ***RMFF*** schedule, while requires no extra processor in the optimal schedule, only to fill part or all of the utilization left by tasks in regions 1 and 2. Note that since $u_2 > 2(1 + (3u_2 + 6u_3) / 10)^{-10} - 1$. The tasks in region 2 can not be scheduled on the processor occupied by tasks in this region. $N = 2N_0 + 2$, and the bound is therefore given by $\Re = 2N_0 / N_0 + 2 / N_0$.

Region 4: If $10 \le N_0 \le 12$, the tasks in regions 1, 2, and 3 are included. Furthermore, there are four more tasks each having a utilization of $(2^{1/5} - 1)$, except the last one with a utilization of $(2^{1/5} - 1) + \varepsilon$, where $\varepsilon$ is an arbitrary small number. These four tasks are placed in one processor in the ***RMFF*** schedule, while requires no extra processor in the optimal schedule, only to fill part of the utilization left by tasks in regions 1, 2, and 3. Note that these tasks do not appear first in the task, rather they follow after the nine tasks in region 3, but before the three tasks each having a utilization of $(2^{1/5} - 1)$. Since $5(2^{1/5} - 1) - 4u_2 < u_2$. The last three tasks in region 3 can not be scheduled on the processor occupied by tasks in this region. $N = 2N_0 + 3$, and the bound is therefore given by $\Re = 2N_0 / N_0 + 2 / N_0$.

This process continues until the largest value of $N$ is found for a given $N_0$, as illustrated by Figure 3. Note that the value $u_i$ is determined by finding the smallest $k$ such that $u_i = (2^{1/k} - 1)$ and $u_i \le 1 - \sum_{l=1}^{i-1} u_l$, for $i \ge 2$.

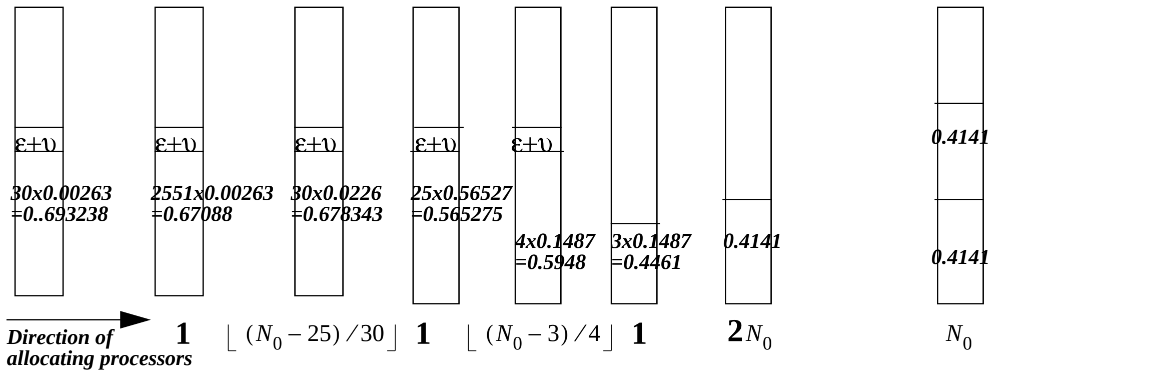
For a given $N_0$, $N = 2N_0 + 1 + \lfloor (N_0 - 3) / 4 \rfloor + 1 + \lfloor (N_0 - 25) / 30 \rfloor + \ldots\ldots$ The bound is given by

$$\Re = \frac{N}{N_0} = \frac{2N_0 + 1 + \lfloor (N_0 - 3) / 4 \rfloor + 1 + \lfloor (N_0 - 25) / 30 \rfloor + \ldots\ldots}{N_0} \approx 2.30. \qquad \textbf{\textit{(E.Q.5)}}$$

For example, given $N_0 = 27$, we construct a set of tasks which, according to ***RMFF Algorithm***, requires $N = 62$ number of processors.

There are $2N_0 = 54$ number of tasks with utilization $u_1 = (2^{1/2} - 1) + \varepsilon$, where $\varepsilon$ is a arbitrary small number. There is one processor occupied by three tasks each with a utilization of $u_2 = (2^{1/5} - 1)$. There are $\lfloor (N_0 - 3) / 4 \rfloor = 6$ number of processors occupied by 6*4 tasks each with a utilization of $(2^{1/5} - 1)$. There is finally a processor occupied by 25 tasks each with a utilization of $u_3$

= 1 - 2*(($2^{1/2}$ - 1) + ε) - ($2^{1/5}$ - 1) - ε. The set of tasks is given as follows. Note that the total number of tasks is 106.



(a) ***RMFF*** Schedule (b) ***Optimal*** Schedule

**Figure 3**: ***RMFF*** vs ***Optimal***

$\tau_i = (u_3, 1)$, for $1 \le i \le 25$.

$\tau_i = (u_2, 1)$ for $26 \le i \le 52$ except $i$ = 29, 33, 37, 41, 45, 49, where $\tau_i = (u_2 + \varepsilon, 1)$.

$\tau_i = (u_1, 1)$ for $53 \le i \le 106$.

According to ***RMFF Algorithm***, The first 25 tasks are scheduled on the first processor. Since $u_2 > 2(1 + 25u_3 / 25)^{-25} - 1$, the 26th task is scheduled on the second processor. The 29th task can not be scheduled on the second processor, since $u_2 + \varepsilon > 2(1 + 4u_3 / 4)^{-4} - 1$. Proceeding in this fashion, the 23 successive tasks occupy 6 processors. The 53th task have to be scheduled on the 8th processor, since $u_1 + \varepsilon > 2(1 + 3u_3 / 34)^{-3} - 1$. The rest of the 53 tasks occupies 53 processors, one task for a processor, since $(2^{1/2} - 1) + \varepsilon > 2(1 + u_1)^{-1} - 1 = 2 / (2^{1/2} + \varepsilon) - 1$. The total number of processors required is thus $N$ = 62. The bound is given by $\Re = \frac{N}{N_0} \approx 2.30$.

**Table 1: Performance of *RMFF* (and also *RMBF)***

| $N_0$ | $\Re$(RMFF) | $N_0$ | $\Re$(RMFF) |
|---|---|---|---|
| 2 | 2 | 10 | 2.30 |
| 3 | 2.33 | 11 | 2.29 |
| 4 | 2.25 | 12 | 2.25 |
| 5 | 2.20 | 13 | 2.31 |
| 6 | 2.33 | 17 | 2.29 |
| 7 | 2.29 | 20 | 2.30 |
| 8 | 2.25 | 27 | 2.30 |
| 9 | 2.22 | 48 | 2.29 |

The exact performance bounds for several given optimal number of processors are given in Table 1. We conjecture that the above formula (E.Q.5) gives the *EXACT* tight bound for ***RMFF Algorithm.*** Q.E.D.

## V. Tight Bound for Rate-Monotonic-Best-Fit

When ***Algorithm RMFF*** schedules a task, it always assigns it to the lowest indexed processor on which the task can be scheduled. This strategy may not be optimal in some cases. For example, the lowest indexed processor on which a task is scheduled may be the one with the largest available utilization among all those busy (non-idle) processors. This processor could have been used to execute a future task with large enough utilization so that it could not be scheduled on any busy processors, had it not been assigned a task with a small utilization earlier on. In order to overcome these likely disadvantages, a new algorithm is designed as follows, which is based on the Best-Fit bin-packing algorithm.

***Algorithm RMBF***: Let the processors be indexed as $P_1$, $P_2$, …, with each initially in the idle state, i.e., with zero utilization. The tasks $\tau_1$, $\tau_2$, …, $\tau_n$, which are ordered according to their non-decreasing periods, will be scheduled in that order. To schedule $\tau_i$, find the least $j$ such that task $\tau_i$, together with all the tasks that have been assigned to processor $P_j$ can be feasibly scheduled according to ***Condition IP*** for a single processor, and $2\,(1 + U_j/k_j)^{-k_j} - 1$ be as small as possible, and assign task $\tau_i$ to $P_j$, where $k_j$ and $U_j$ are the number of tasks already assigned to processor $P_j$ and the total utilization of the $k_j$ tasks, respectively, and $u_i$ is the utilization of task $\tau_i$.

Surprisingly, even with this modification in assigning tasks to processors, the ***RMBF Algorithm*** does not outperform ***Algorithm RMFF*** in the worst-case, as shown by Theorem 5.1 and Theorem 5.2. Before we prove the tight bound for ***RMBF***, the following definition is needed, which is key to the proof of Theorem 5.1.

***Definition 1***: For all the processors required to schedule a given set of tasks by the ***RMBF Algorithm***, they are divided into two types of processors:

Type (I): For all the tasks $\tau_1, \tau_2, \ldots, \tau_m$ with utilizations $u_1, u_2, \ldots, u_m$ that were assigned to a processor $P_x$ in the completed ***RMBF*** schedule, there exists at least one task $\tau_i$ with $i \geq 2$ that was assigned to $P_x$, not because it could not be assigned on any processor $P_y$ with lower index, i.e., $y < x$, but because $2\,(1 + (\sum_{l=1}^{i-1} u_l)/(i-1))^{-(i-1)} - 1 < 2\,(1 + (\sum_{l=1}^{n_y} u_l)/n_y)^{-n_y} - 1$, where $n_y$ is the number of tasks assigned to processor $P_y$. Processor $P_x$ is called a Type (I) processor. Such a task $\tau_i$ is, for convenience, referred to as a task with Type (I) property.

Type (II): They consist of all the processors that do not belong to Type (I).

***Lemma 5.1***: For ***Algorithm RMBF***, the following properties hold:

(1) No task is assigned to an idle processor unless it can not be assigned in any non-idle processor.

***Proof***: For ***Algorithm RMBF***, properties (1) is true according to its definition. Q.E.D.

***Lemma 5.2***: If $m$ tasks can not be feasibly scheduled on $m-1$ processors according to the ***RMBF Algorithm***, then the utilization factor of the set of tasks is greater than $m(2^{1/2}-1)$.

***Proof***: The proof of this lemma is similar to that of Lemma 4.1. Q.E.D.

The two lemmas given below follow directly from Lemma 4.2 and Lemma 4.3.

***Lemma 5.3***: In the completed ***RMBF*** schedule, among all processors of Type (II), to each of which two tasks are assigned, there is at most one processor for which the total utilization factor of the set of the two tasks is less than or equal to $2(2^{1/3}-1)$.

***Lemma 5.4***: In the completed ***RMBF*** schedule, among all processors of Type (II), to each of which $n$ tasks are assigned, there is at most one processor for which the total utilization factor of the set of the $n$ tasks is less than or equal to $n(2^{1/(n+1)}-1)$. $\lim_{n\to\infty} n(2^{1/(n+1)}-1) = ln2$.

***Lemma 5.5***: In the completed ***RMBF*** schedule, if the second task on any of the Type (I) processors has Type (I) property, then the first task on that processor has a utilization greater than $(2^{1/2}-1)$.

***Proof***: Let $\tau_{k,1}$ and $\tau_{k,2}$ be the first and second tasks assigned to processor $P_k$ of Type (I), and $P_y$, with $y < k$, is one of the processors on which $\tau_{k2}$ could have been scheduled, but $2(1+u_{k,1})^{-1}-1 < 2\left(1+\left(\sum_{l=1}^{n_y} u_{y,l}\right)/n_y\right)^{-n_y} - 1$, where $n_y$ is the number of tasks assigned to processor $P_y$, and where $u_{x,l}$ is the utilization of task $\tau_{x,l}$.

Since $u_{k,1} > 2\left(1+\left(\sum_{l=1}^{n_y} u_{y,l}\right)/n_y\right)^{-n_y} - 1$ (note that this is true even though $\tau_{k,1}$ is assigned to processor $P_k$ before some of tasks among the $n_y$ tasks are assigned to processor $P_y$), $u_{k,1} > 2\left(1+\left(\sum_{l=1}^{n_y} u_{y,l}\right)/n_y\right)^{-n_y} - 1 > 2(1+u_{k,1})^{-1}-1$. Therefore $u_{k,1} > (2^{1/2}-1)$. Q.E.D.

***Lemma 5.6***: In the completed ***RMBF*** schedule, if the $m$th task on any of the Type (I) processors has Type (I) property, where $m \geq 3$, then the total utilization of the first $(m-1)$ tasks on that processor is greater than $(m-1)(2^{1/m}-1)$.

The proof of this lemma is given in the appendix.

The following lemma is key to the proof of Theorem 5.1.

***Lemma 5.7***: In the completed ***RMBF*** schedule, among the processors of Type (I) on which the second task has Type (I) property, there are at most three of them, each of which has a total utilization less than $2(2^{1/3}-1)$.

***Proof***: This lemma is proven by contradiction. Let $P_i$, $P_j$, $P_k$, and $P_l$ be the four processors, each of which has a total utilization less than $2(2^{1/3}-1)$ with $i < j < k < l$, i.e.,

$$\sum_{x=1}^{n_i} u_{i,x} < 2(2^{1/3}-1)$$
$$\sum_{x=1}^{n_j} u_{j,x} < 2(2^{1/3}-1)$$
$$\sum_{x=1}^{n_k} u_{k,x} < 2(2^{1/3}-1)$$
$$\sum_{x=1}^{n_l} u_{l,x} < 2(2^{1/3}-1)$$

where $n_i \geq 2$, $n_j \geq 2$, $n_k \geq 2$, and $n_l \geq 2$ are the number of tasks assigned to processors $P_i$, $P_j$, $P_k$, and $P_l$, respectively.

Let's define $u_{i,1}$ and $u_{i,2}$ to be the utilizations of the first task $\tau_{i,1}$ and second tasks $\tau_{i,2}$ assigned to processor $P_i$, $u_{j,1}$ and $u_{j,2}$ to be the utilizations of the first task $\tau_{j,1}$ and second tasks $\tau_{j,2}$ assigned to processor $P_j$. $u_{k,1}$ and $u_{k,2}$, $u_{l,1}$ and $u_{l,2}$ are similarly defined. We further assume that $n_y$ is the number of tasks which have been assigned to processor $P_i$, when the second task on processor $P_j$ is assigned. Note that $i < j$ and $1 \leq n_y \leq n_j$.

There are three cases to consider.

Case 1: Tasks $\tau_{j,1}$ and $\tau_{j,2}$ are assigned to processor $P_j$ after task $\tau_{i,2}$ is assigned to processor $P_i$. Since task $\tau_{j,2}$ is a Type (I) task, the following inequality must hold

$$2(1 + u_{j,1})^{-1} - 1 < 2\left(1 + \left(\sum_{x=1}^{n_y} u_{i,x}\right)/n_y\right)^{-n_y} - 1$$

Note that $n_y \geq 2$, i.e., other tasks may have been assigned to processor $P_i$ after task $\tau_{i,2}$ but before $\tau_{j,1}$ is assigned to processor $P_j$.

Since $2\left(1 + \left(\sum_{x=1}^{n_y} u_{i,x}\right)/n_y\right)^{-n_y} - 1 \leq 2(1 + (u_{i,1} + u_{i,2})/2)^{-2} - 1 < 2(1 + u_{i,1}/2)^{-2} - 1$,

$2(1 + u_{j,1})^{-1} - 1 < 2(1 + u_{i,1}/2)^{-2} - 1$, i.e., $1 + u_{j,1} > (1 + u_{i,1}/2)^2$.

Case 2: Tasks $\tau_{j,1}$ and $\tau_{j,2}$ are assigned to processor $P_j$ after task $\tau_{i,1}$ is assigned to processor $P_i$ but before task $\tau_{i,2}$ is assigned to processor $P_i$.

This case is impossible with ***RMBF*** scheduling. Since $\sum_{x=1}^{n_i} u_{i,x} < 2(2^{1/3}-1)$ and $u_{i,1} > (2^{1/2}-1)$ according to Lemma 5.5, $u_{i,2} < 2(2^{1/3}-1) - (2^{1/2}-1) \approx 0.1056$. Since task $\tau_{j,2}$ is assigned to processor $P_j$ before task $\tau_{i,2}$ is assigned to processor $P_i$, and task $\tau_{j,2}$ is a Type (I) task, $2(1 + u_{i,1})^{-1} - 1 > 2(1 + u_{j,1})^{-1} - 1$, i.e.,

$u_{i,1} < u_{j,1}$. ***(E.Q.6)***

Since task $\tau_{i,2}$ is also a Type (I) task, it must be true according to the definition that

$2(1 + u_{i,1})^{-1} - 1 < 2\left(1 + \left(\sum_{x=1}^{n_z} u_{j,x}\right)/n_z\right)^{-n_z} - 1$, where $n_z$ is the number of tasks that have been assigned to processor $P_j$ after task $\tau_{j,2}$, but before task $\tau_{i,2}$ is assigned to processor $P_i$. Note that it is conceivable that other tasks may have been assigned to processor $P_j$ after task $\tau_{j,2}$ but before task $\tau_{i,2}$ is assigned to processor $P_i$.

Since $2(1 + u_{i,1})^{-1} - 1 < 2\left(1 + \left(\sum_{x=1}^{n_z} u_{j,x}\right)/n_z\right)^{-n_z} - 1 < 2(1 + u_{j,1})^{-1} - 1$, $u_{i,1} > u_{j,1}$.

This is a contradiction to equation (E.Q.6).

Case 3: Task $\tau_{j,1}$ is assigned to processor $P_j$ after task $\tau_{i,1}$ is assigned to processor $P_i$, and task $\tau_{j,2}$ is assigned to processor $P_j$ after task $\tau_{i,2}$ is assigned to processor $P_i$. Since task $\tau_{j,2}$ is a Type (I) task, the following inequality must hold

$$2(1 + u_{j,1})^{-1} - 1 < 2\left(1 + \left(\sum_{x=1}^{n_y} u_{i,x}\right)/n_y\right)^{-n_y} - 1$$

Note that $n_y \geq 2$, i.e., other tasks may have been assigned to processor $P_i$ after task $\tau_{i,2}$ but before $\tau_{j,2}$ is assigned to processor $P_j$.

Since $2\left(1 + \left(\sum_{x=1}^{n_y} u_{i,x}\right)/n_y\right)^{-n_y} - 1 \leq 2(1 + (u_{i,1} + u_{i,2})/2)^{-2} - 1 < 2(1 + u_{i,1}/2)^{-2} - 1$,

$2(1 + u_{j,1})^{-1} - 1 < 2(1 + u_{i,1}/2)^{-2} - 1$, i.e., $1 + u_{j,1} > (1 + u_{i,1}/2)^2$.

Therefore for processors $P_i$ and $P_j$, we have

$$1 + u_{j,1} > (1 + u_{i,1}/2)^2. \qquad \textbf{\textit{(E.Q.7)}}$$

For the tasks assigned on processors $P_j$ and $P_k$, and $P_k$ and $P_l$, it can be similarly proven that

$$1 + u_{k,1} > (1 + u_{j,1}/2)^2 \qquad \textbf{\textit{(E.Q.8)}}$$

$$1 + u_{l,1} > (1 + u_{k,1}/2)^2 \qquad \textbf{\textit{(E.Q.9)}}$$

Summing up equations (E.Q.7), (E.Q.8), and (E.Q.9) yields $u_{l,1} > (u_{i,1}^2 + u_{j,1}^2 + u_{k,1}^2)/4 + u_{i,1}$. Since $u_{i,1} > (2^{1/2}-1)$, $u_{j,1} > (2^{1/2}-1)$, and $u_{k,1} > (2^{1/2}-1)$ according to Lemma 5.5, $u_{l1} > 3(2^{1/2}-1)^2/4 + (2^{1/2}-1) = 0.5429 > 2(2^{1/3}-1)$. This results in a contradiction to the assumption that $\sum_{x=1}^{n_l} u_{l,x} < 2(2^{1/3}-1)$. Q.E.D.

***Theorem 5.1***: Let $N$ be the number of processors required to feasibly schedule a set of tasks by the ***RMBF Algorithm***, and $N_0$ the minimum number of processors required to feasibly schedule the same set of tasks. Then $\lim_{N_0 \to \infty} N/N_0 \leq 2 + (3 - 2^{3/2})/a \approx 2.33$, where $a = 2(2^{1/3} - 1)$.

In order to prove the above bound, we define a function that maps the utilization of tasks into the real interval [0, 1] as it is done in the previous section. The function is the same as the one used for ***RMFF Algorithm***.

For a processor $P_j$, its deficiency $\delta_j$ and its coarseness $\alpha_j$ are similarly defined as those for ***RMFF Algorithm***. Also note that Lemma 4.6, Lemma 4.7, and Lemma 4.8 also hold for those processors of Type (II) in the ***RMBF*** schedule. The following lemma is also true.

***Lemma 5.8***: If a processor is assigned a number of tasks $\tau_1, \tau_2, \ldots, \tau_m$, with utilizations $u_1, u_2, \ldots, u_m$, then $\sum_{i=1}^{m} f(u_i) \leq 1/a$, where $a = 2(2^{1/3} - 1)$.

Proof of Theorem 5.1: Let $\Sigma = \{\tau_1, \tau_2, \ldots, \tau_m\}$ be a set of $m$ tasks, with their utilizations $u_1, u_2, \ldots, u_m$ respectively, and $\varpi = \sum_{i=1}^{m} f(u_i)$. By Lemma 5.8, $\varpi \leq N_0 / a$, where $a = 2(2^{1/3} - 1)$.

Suppose that among the $N$ number of processors used by ***RMBF Algorithm*** to schedule a given set $\Sigma$ of tasks, $M_1$ of them belongs to processors of Type (I). Since all processors of Type (I) must be assigned at least two tasks, there exists for each processor at least an number $m$ with $m \geq 2$ such that the $m$th task is a Type (I) task. For all the processors of Type (I) on each of which the $m$th task is a Type (I) task with $m \geq 3$, $\sum_j f(u_j) > 1$ since $\sum_j u_j > 2(2^{1/3} - 1)$ according to Lemma 5.6.

When $m = 2$, there are at most three of them, each of which has a total utilization less than $2(2^{1/3} - 1)$. Therefore, for all the processors of Type (I), there are at most three processors whose $\sum_j f(u_j)$ is less than $1$ in the ***RMBF*** schedule.

Now let $L = n_1 + n_2 + n_3$ be defined similarly as in Section IV, except that they are for processors of Type (II). All the results derived in Section IV are applicable to the set of Type (II) processors in the ***RMBF*** schedule

Now we are ready to find out the relationship between $N$ and $N_0$.

$$\begin{aligned}\varpi = \sum_{i=1}^{m} f(u_i) &\geq (N - L - 3) + n_1 (2^{1/2} - 1) / a + n_3 - 4/3 \\ &= N - n_1 - n_2 - n_3 + n_1 (2^{1/2} - 1) / a + n_3 - 13/3 \\ &\geq N - 2N_0(1 - (2^{1/2} - 1)/a) - n_2 - 13/3, \text{ where } a = 2(2^{1/3} - 1).\end{aligned}$$

Since $\varpi \leq N_0 / a$, $N_0 / a \geq N - 2N_0(1 - (2^{1/2} - 1)/a) - n_2 - 13/3$

Therefore, $N / N_0 \leq (2a + 1 - 2(2^{1/2} - 1)) / a + 16/(3N_0)$.

$\lim_{N_0 \to \infty} N/N_0 \leq (2a + 1 - 2(2^{1/2} - 1)) / a \approx 2.33$. Q.E.D.

***Theorem 5.2***: Let $N$ be the number of processors required to feasibly schedule a set of tasks by ***RMBF Algorithm***, and $N_0$ the minimum number of processors required to feasibly schedule the same set of tasks. Then $\lim_{N_0 \to \infty} N/N_0 \geq 2.3$.

***Proof***: The proof of Theorem 4.2 is applicable to the proof of this theorem. Q.E.D.

## VI. Concluding Remarks

In this paper, we are motivated by the increasingly important role played by the rate-monotonic algorithm in designing predictable real-time systems. The problem of scheduling a set of periodic tasks on a multiprocessor using a fixed priority assignment scheme is studied, and the performance of the first two scheduling heuristics used to solve the problem is revisited. The worst-case performance of these algorithms is studied since task deadlines in a hard real-time system have to be guaranteed even in the worst cases. The worst-case performance bounds are tightened up to be 2.33 for *RMFF* and 2.67 for *RMNF*. A new scheduling algorithm — *RMBF* was proposed as an alternative to *RMFF*, and it also has a tight worst-case bound of 2.33. The analytic results presented here are the few ones on scheduling periodic tasks on multiprocessors.

Since these three algorithms require that tasks are ordered according to their non-decreasing periods, they are static algorithms. These algorithms obviously are not applicable in situations where the scheduling decisions have to be made dynamically, since the period of an incoming task may be shorter than some of the tasks already assigned to some processors. Therefore, dynamic algorithms need to be developed. We are currently investigating the performance of several dynamic algorithms.

## Appendix

Before we prove Lemma 4.3, we need to prove the following lemma.

***Lemma 4.9***: If tasks are assigned to the processors according to the ***RMFF Algorithm***, among all processors to each of which one task is assigned, there is at most one processor for which the utilization factor of the task is less than or equal to $(2^{1/2}-1)$.

***Proof:*** This lemma is proven by contradiction. The contrary is supposed to be true, i.e., there are at least two processors, each of which has utilization less than or equal to $(2^{1/2}-1)$. Let $\tau_j$ be the task with utilization equal to $u_j$, that is assigned to processor $P_j$, and $\tau_k$ be the task with utilization equal to $u_k$, that is assigned to processor $P_k$ with $j < k$, such that

$$u_j \le (2^{1/2}-1) \text{ and } u_k \le (2^{1/2}-1)$$

Summing up these two inequalities yields

$$u_k + u_k \le 2(2^{1/2}-1)$$

This implies that tasks $\tau_j$ and $\tau_k$ are assigned on a single processor, which is a contradiction to the assumption. Q.E.D.

***Lemma 4.3***: If tasks are assigned to the processors according to the ***RMFF Algorithm***, among all processors to each of which $n \ge 1$ tasks are assigned, there is at most one processor for which the utilization factor of the set of the $n$ tasks is less than $n(2^{1/(n+1)}-1)$.

$$\lim_{n \to \infty} n\,(2^{1/(n+1)} - 1) = ln2$$

***Proof:*** This lemma holds when $n$ is equal to 1or 2 according to Lemma 4.9 and Lemma 4.2. Now suppose that the lemma holds for $n \le k$. The lemma is proven to be true for $n = k + 1$ by contradiction. Let $n = k + 1$, and $P_i$ and $P_j$ with $i < j$ be the two processors on each of which exactly $n$ tasks are assigned, such that the total utilization of the $n$ tasks on each processor satisfies

$$\sum_{m=1}^{k+1} u_{i,m} < (k+1)(2^{1/(k+2)}-1) \qquad \textbf{\textit{(E.Q.10)}}$$

and

$\sum_{m=1}^{k+1} u_{j,m} < (k + 1)(2^{1/(k+2)}-1)$. ***(E.Q.11)***

respectively, where $u_{i,m}$ denotes the utilization of the $m$th task assigned on processor $i$. Since processors $P_i$ and $P_j$ are each assigned $n = k + 1$ tasks, we must have

$u_{i,k+1} \le 2(1 + \sum_{m=1}^{k} u_{i,m}/k)^{-k} - 1$

and

$u_{j,k+1} \le 2(1 + \sum_{m=1}^{k} u_{j,m}/k)^{-k} - 1$

Assume that $\Delta^i = \sum_{m=1}^{k+1} u_{i,m}$ and $\Delta^j = \sum_{m=1}^{k+1} u_{j,m}$. Among the $n$ tasks which are assigned to processor $P_j$, task $\tau_{j,x}$ is the first task that is assigned to processor $P_j$ immediately after task $\tau_{i,k+1}$ was assigned to processor $P_i$, $1 \le x \le k+1$. We will consider the boundary condition where task $\tau_{j,k+1}$ is assigned to processor $P_j$ before task $\tau_{i,k+1}$ is assigned to processor $P_i$.

Case 1: $1 \le x \le k+1$.

For $x \le z \le k+1$, since $\tau_{j,z}$ can not be scheduled on processor $P_i$ even after $\tau_{i,k+1}$ has been scheduled on $P_i$, we must have

$u_{j,z} > 2(1 + \Delta^i/(k+1))^{-(k+1)} - 1$

Since $\Delta^i = \sum_{m=1}^{k+1} u_{i,m} < (k + 1)(2^{1/(k+2)}-1)$ from equation (E.Q.10),

$u_{j,z} > 2(1 + 2^{1/(k+2)}-1)^{-(k+1)} - 1 = 2^{1/(k+2)} - 1$.

For $1 \le z < x$, since $\tau_{j,z}$ can not be scheduled on processor $P_i$ before $\tau_{i,k+1}$ is scheduled on $P_i$, we must have

$u_{j,z} > 2(1 + \Delta^i_y/y)^{-y} - 1$, for some $y \le k$ and $\Delta^i_y = \sum_{m=1}^{y} u_{i,m}$.

Since $2(1 + \Delta^i_y/y)^{-y} - 1 \ge 2(1 + \Delta^i_{k+1}/(k+1))^{-(k+1)} - 1$, and $\Delta^i = \sum_{m=1}^{k+1} u_{i,m} < (k + 1)(2^{1/(k+2)}-1)$ from equation (E.Q.10),

$u_{j,z} > 2^{1/(k+2)} - 1$

$\Delta^j = \sum_{m=1}^{x-1} u_{j,m} + \sum_{m=x}^{k+1} u_{j,m} > (k+1)(2^{1/(k+2)}-1)$,

which is a contradiction to equation (E.Q.11).

Case 2: The boundary condition where task $\tau_{j,k+1}$ is assigned to processor $P_j$ before task $\tau_{i,k+1}$ is assigned to processor $P_i$.

For $1 \le z \le k+1$, since $\tau_{j,z}$ can not be scheduled on processor $P_i$ before $\tau_{i,k+1}$ is scheduled on $P_i$, we must have

$u_{j,z} > 2(1 + \Delta^i_y/y)^{-y} - 1$, for some $y \le k$ and $\Delta^i_y = \sum_{m=1}^{y} u_{i,m}$.

Since $2(1 + \Delta^i_y/y)^{-y} - 1 \ge 2(1 + \Delta^i_{k+1}/(k+1))^{-(k+1)} - 1$, and $\Delta^i = \sum_{m=1}^{k+1} u_{i,m} < (k + 1)(2^{1/(k+2)}-1)$ from equation (E.Q.10),

$u_{j,z} > 2^{1/(k+2)} - 1$

$\Delta^j = \sum_{m=1}^{k+1} u_{j,m} > (k+1)(2^{1/(k+2)}-1)$,

which is a contradiction to equation (E.Q.11). Q.E.D.

***Lemma 5.6***: In the completed ***RMBF*** schedule, if the $m$th task on any of the Type (I) processors has Type (I) property, where $m \ge 3$, then the total utilization of the first $(m-1)$ tasks on that processor is greater than $(m-1)(2^{1/m}-1)$.

***Proof***: Let $\tau_{k,1}, \tau_{k,2}, \ldots, \tau_{k,m-1}$ be the tasks that were assigned a processor $P_k$ of Type (I), and

$P_y$, with $y < k$, is one of the processors on which $\tau_m$ could have been scheduled, but $2\left(1 + \left(\sum_{l=1}^{m-1} u_{k,l}\right)/(m-1)\right)^{-(m-1)} - 1 < 2\left(1 + \left(\sum_{l=1}^{n_y} u_{y,l}\right)/n_y\right)^{-n_y} - 1$, where $n_y$ is the number of tasks assigned to processor $P_y$, and where $u_{x,l}$ is the utilization of task $\tau_{x,l}$ on processor $P_x$.

Since $u_{k,i} > 2\left(1 + \left(\sum_{l=1}^{n_y} u_{y,l}\right)/n_y\right)^{-n_y} - 1$ (note that this is true even though $\tau_{k,i}$ is assigned to processor $P_k$ before some of tasks among the $n_y$ tasks are assigned to processor $P_y$), for $1 \le i \le m-1$, $u_{k,i} > 2\left(1 + \left(\sum_{l=1}^{n_y} u_{y,l}\right)/n_y\right)^{-n_y} - 1 > 2\left(1 + \left(\sum_{l=1}^{m-1} u_{k,l}\right)/(m-1)\right)^{-(m-1)} - 1$. Summing up these $(m - 1)$ inequalities yields

$$\sum_{j=1}^{m-1} u_{k,j} > 2(m-1)\left(1 + \left(\sum_{l=1}^{m-1} u_{k,l}\right)/(m-1)\right)^{-(m-1)} - (m-1).$$

Solving the above equation yields

$$\sum_{j=1}^{m-1} u_{k,j} > (m-1)(2^{1/m}-1).$$ Q.E.D.

## References

[1] E.G. COFFMAN, JR. (ED.), *Computer and Job Shop Scheduling Theory,* New York: Wiley, 1975.

[2] E.G. COFFMAN, JR., M.R. GAREY, AND D.S. JOHNSON, "Approximate Algorithms for Bin Packing - An Updated Survey," In *Algorithm Design for Computer System Design*, pp. 49-106, G. AUSIELLO, M. LUCERTINIT, and P. SERAFINI (Eds), Springer-Verlag, New York, 1985.

[3] S. DAVARI AND S.K. DHALL, "An On Line Algorithm for Real-Time Tasks Allocation," *IEEE Real-Time Systems Symposium*, 194-200 (1986).

[4] S. DAVARI AND S.K. DHALL, "On a Periodic Real-Time Task Allocation Problem," *Proc. of 19th Annual International Conference on System Sciences*, 133-141 (1986).

[5] S.K. DHALL AND C.L. LIU, "On a Real-Time Scheduling Problem," *Operations Research* **26**, 127-140 (1978).

[6] M.R. GAREY AND D.S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-completeness*, W.H. Freeman and Company, NY, 1978.

[7] D.S. JOHNSON, *Near-Optimal Bin Packing Algorithms*, Doctoral Thesis, MIT, 1973

[8] J. LEHOCZKY, L. SHA, AND Y. DING, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," *IEEE Real-Time Symposium*, 166-171 (1989).

[9] J.P. LEHOCZKY, L. SHA, AND J.K. STROSNIDER. "Enhanced Aperiodic Responsiveness in Hard Real-time Environments," *IEEE Real-Time Systems Symposium*, 261-270 (1987).

[10] J.P. LEHOCZKY AND S. RAMOS-THUEL. "An Optimal ALgorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems," *IEEE Real-Time Systems Symposium*, 110-123 (1992).

[11] 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).

[12] C.L. LIU AND J. LAYLAND, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *J. Assoc. Comput. Machinery* **10(1)**, 174-189 (1973).

[13] J.W.S. LIU, K.-J. LIN, AND S. NATARAJAN. "Scheduling Real-time, Periodic Jobs Using Imprecise Results," *IEEE Real-Time Systems Symposium*, 252-260 (1987).

[14] J.W.S. LIU, K.-J. LIN, W.K. SHIH, A.C. YU, J.Y. CHUNG AND W. ZHAO. "Algorithms for Scheduling Imprecise Computations," *Computer*, 58-68 (May 1991).

[15] K. RAMAMRITHAM. "Allocation and Scheduling of Complex Periodic Tasks," *International Conference on Distributed Computing Systems*, May 1990.

[16] S. RAMOS-THUEL AND J.K. STROSNIDER. "The Transient Server Approach to Scheduling Time-Critical Recovery Operations," *IEEE Real-Time Systems Symposium*, 286-295 (1991).

[17] P. SERLIN, "Scheduling of Time Critical Processes," *Proceedings of the Spring Joint Computers Conference* **40**, 925-932 (1972).

[18] L. SHA, J.P. LEHOCZKY, AND R. RAJKUMAR. "Solutions for Some Practical Problems in Prioritized Preemptive Scheduling," *IEEE Real-Time Systems Symposium*, 181-191 (1986).

[19] L. SHA, R. RAJKUMAR, J.P. LEHOCZKY, AND K. RAMAMRITHAM. "Mode Change Protocols for Priority-Driven Preemptive Scheduling," *Journal of Real-Time Systems* **1(3)**, 244-264 (1989)

[20] L. SHA, R. RAJKUMAR, AND J.P. LEHOCZKY. "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Transactions on Computers* **39(9)**, 1175-1185 (1990).

[21] W-K. SHIH, J.W.S. LIU, AND J-Y CHUNG. "Fast Algorithms for Scheduling Imprecise Computations," *IEEE Real-Time Systems Symposium*, 12-19 (1989).

[22] B. SPRUNT, L. SHA, AND J.P. LEHOCZKY. "Aperiodic Task Scheduling for Hard Real-time Systems," *Journal of Real-Time Systems* **1**, 27-60 (1989).

[23] K.W. TINDELL, A. BURNS, AND A.J. WELLINGS. "Mode Change in Priority Pre-emptively Scheduled Systems," *IEEE Real-Time Systems Symposium*, 100-109 (1992).