Preemptive Scheduling of Periodic Tasks on Multiprocessor: Dynamic Algorithms and Their Performance

Yingfeng Oh and Sang H. Son

Technical Report No. CS-93-26
May 24, 1993

Preemptive Scheduling of Periodic Tasks on Multiprocessor: Dynamic Algorithms and Their Performance

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

Abstract

In this paper, the problem of preemptively scheduling a set of periodic tasks on a multiprocessor is considered. Three dynamic algorithms are proposed, and their performance is studied. These algorithms are Rate-Monotonic-Next-Fit-WC (***RMNF-WC***), Rate-Monotonic-First-Fit-WC (***RMFF-WC***), and Rate-Monotonic-Best-Fit-WC (***RMBF-WC***), and their worst-case performance is shown to be tightly bounded by 2.88, 2.33, and 2.33, respectively. The major contributions of this papers are (1) These algorithms are the few truly dynamic algorithms for scheduling periodic tasks on a multiprocessor system, and they are the few algorithms, the worst-case performance of which is investigated. (2) The worst-case performance bound is shown to be tight. (3) The worst-case performance bound of ***RMFF-WC*** is as good as that of its static counterpart — ***RMFF*** studied by Dhall and Liu. (4) A new scheduling heuristic — ***RMBF-WC*** is proposed and its worst-case performance investigated.

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[10], and Serlin[12]. In the case of fixed priority assignment, the rate-monotonic algorithm [10] or [12] was proven to be optimal. In the case of dynamic priority assignment, the earliest deadline first (*EDF*) algorithm [10] was 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. 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 within the framework

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

of rate-monotonic scheduling, for example, the scheduling of tasks which need to be synchronized, the scheduling of real-time tasks that are “imprecise”, the scheduling of aperiodic and sporadic tasks, and the scheduling support to overcome transient overload.

In this paper, we consider the problem of scheduling a set of periodic tasks on a multiprocessor system. Since this problem is proven to be *NP-hard* [9], for practical purpose, scheduling heuristics need to be devised to obtain approximate solutions. Although there may be potentially numerous scheduling heuristics to solve this problem, we focus our studies on a particular class of scheduling heuristics, which uses rate-monotonic algorithm to schedule the set (or subset with respect to the whole task set) of tasks assigned on each individual processor. This approach was also pursued by a number of other researchers [5] [3] [4]. There are a number of reasons that justify this study: First, 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. Secondly, rate-monotonic algorithm is optimal for fixed-priority assignment of periodic tasks on a processor. The reason to use fixed-priority assignment is for practical purposes, such as the ease of implementation and minimal scheduling overhead involved. 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 sporadic tasks and soft-deadline tasks, and the scheduling of tasks which need to be synchronized or have resource requirements.

Dhall and Liu [5] first proposed two heuristic algorithms to solve this problem, and analyzed their performance. These two heuristics are called the Rate-Monotonic-Next-Fit (*RMNF*) algorithm and Rate-Monotonic-First-Fit (*RMFF*) algorithm. These two algorithms are based on the assumption that tasks are assigned to processors in the order of non-decreasing task periods. The performance of *RMNF* and *RMFF* was proven to be upper bounded by 2.76 and 2.23, and lower bounded by 2.4 and 2.0, respectively. Recently, Oh and Son [11] proved that the performance of *RMNF* was tightly bounded by 2.76, and *RMFF* by 2.33, correcting an error existing in [5][1]. These two algorithms, however, require apriori knowledge about the tasks to be scheduled, and hence they are static algorithms.

Davari and Dhall [3] [4] later studied two other scheduling heuristics: the First-Fit-Decreasing-Utilization-Factor (*FFDUF*) and *NEXT-FIT-M*. The *FFDUF* algorithm sorts the set of tasks in non-increasing order of utilization factor and assigns the tasks to processors in that order. The *NEXT-FIT-M* algorithm 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

1. Readers can convince themselves of the existence of errors in [5] by reading theorem 4.2, since the worst-case examples given in the theorem are also the worst-case examples for *RMFF*. If interested, see [11] for details.

exclusively. The 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 function of the pre-selected number *M*. The *FFDUF* is obviously an static algorithm. In the general sense, the *NEXT-FIT-M* algorithm is a dynamic algorithm, but its performance depends on the pre-selection of *M* and henceforth S_M, where S_M is a decreasing function of *M*, e.g., $S_M = 2.34$ for $M = 4$, and $S_M = 2.28$ for $M \rightarrow \infty$.

Since real-time systems often operate in dynamic and complex environments, many scheduling decisions have to be made dynamically, and hence dynamic scheduling algorithms are essential in implementing these decisions. In the following, we propose three dynamic algorithms to solve the same scheduling problem. These three scheduling algorithms are all based on some bin-packing heuristics, but also differ significantly from them in some other aspects. The reason to choose bin-packing heuristics is because assigning tasks on processors bears many similarity to packing items into bins. 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.

We first study two dynamic scheduling algorithms — Rate-Monotonic-Next-Fit-WC (worst-case) or *RMNF-WC*, and Rate-Monotonic-First-Fit-WC (worst-case) or *RMFF-WC*. These two algorithms are based on bin-packing heuristics, and Liu and Layland's worst-case bounds are used as the schedulability condition. *RMNF-WC* is studied because of its simplicity, while the reason to study *RMFF-WC* is because First-Fit is one of the best heuristics for bin-packing. The way that these two algorithms are so called is to distinguish them from the other two algorithms — *RMNF* and *RMFF* studied by Dhall and Liu [5]. The key difference between these two algorithms — *RMNF-WC* and *RMFF-WC* and *RMNF* and *RMFF* is that *RMNF-WC* and *RMFF-WC* are truly dynamic algorithms, while *RMNF* and *RMFF* are static algorithms. The worst-case performance of *RMFF-WC* is shown to be tightly bounded by 2.33, which is surprisingly the same performance bound offered by *RMFF* and to some extent, by *NEXT-FIT-M*.

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

This paper is organized as follows. In the next section, the scheduling problem is formally defined. The performance of *RMNF-WC* is proven to be tightly bounded by $2 / (\ln 2)$ in Section III. The *RMFF-WC* algorithm is presented, and its performance analyzed in Section IV, while the performance of *RMBF-WC* is given in Section V. Finally, we conclude in Section VI and indicate the remaining problems.

II. Problem Definition

The problem of scheduling a set of periodic tasks on a multiprocessor is defined as follows: Given a set of n tasks $\Sigma = \{\tau_1, \tau_2, \ldots, \tau_n\}$, where each task τ_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 deadline of a task is assumed to be equal to its period, and the tasks are independent. The preemptive scheduling discipline is also assumed.

To solve this problem, a heuristic approach which consists of two steps is usually adopted: a heuristic algorithm is first employed to assign tasks to processors, and then 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 much 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 [10]. 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***. 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 τ_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. Another schedulability condition, which was given by Lehoczky et al [8], takes into account both the computation time and the period of a task when a task is scheduled. It is called ***Condition IFF*** (*IF and only iF*) since it is a sufficient and necessary condition.

Condition IFF: Given a set of periodic tasks $\Sigma = \{\tau_1, \tau_2, \ldots, \tau_n\}$,

1. τ_i can be scheduled for all task phasings using the rate monotonic algorithm if and only if $L_i = min_{\{t \in S_i\}}((W_i(t))/t) \le 1$;
2. The entire task set is schedulable for all task phasings using the rate monotonic algorithm if and only if $L = max_{\{1 \le i \le m\}} L_i \le 1$;

where $S_i = \{kT_j \mid j = 1, \ldots, i;\ k = 1, \ldots, \lceil T_i/T_j \rceil\}$, $W_i(t) = \sum_{j=1}^{i} C_j \lceil t/T_j \rceil$, $L_i(t) = W_i(t)/t$, $L_i = min_{\{t \in S_i\}} L_i(t)$.

For scheduling a set of periodic tasks in the order of non-decreasing periods on a single processor, the following relation obviously holds: ***Condition WC*** $\subset$ ***Condition IP*** $\subset$ ***Condition IFF***. However, this relation does not imply that using the same heuristic for assigning tasks on processors, but under different schedulability conditions, similar relation on the number of processors allocated in the worst case will also hold. In the case of ***Condition WC*** vs ***Condition IP***, the worst-case performance bounds for using the different heuristics exhibit different relationships. In some other cases, trying to maximize the utilization of a processor locally does not automatically lead to the minimization of the number of processors used. As an example, ***RMBF-WC*** tries to maximize the utilization of a processor, yet the overall performance of ***RMBF-WC*** is no better than that of ***RMFF-WC***. It is, therefore, quite interesting to investigate how good each bin-packing heuristic, combined with different schedulability condition, perform in the worst-case. Among a number of bin-packing heuristics, Next-Fit, First-Fit, and Best-Fit are of particular interest to not only computer scientists, but also researchers in other fields.

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 lth task that is assigned on the xth processor. $u_{x,l}$ denotes the utilization of task $\tau_{x,l}$. τ_i is used to denote the ith task where there is no confusion. u_i denotes the utilization of the ith task on a processor or in a task set. $\tau = (x, y)$ characterizes a task τ, where x and y are the computation time and the period of task τ.

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

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

Algorithm RMNF-WC:

1. Set $i = j = 1$. /* i denotes the ith task, j the number of processors allocated */
2. Assign task τ_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 WC***. If not, assign task τ_i to P_{j+1} and set $j = j + 1$.
3. If $i < n$, then set $i = i + 1$ and go to step 2 else stop.

When the algorithm finishes, the value in j is the number of processors required to execute a given task set. In order to obtain the tight bound of its worst-case performance, we first prove its upper bound, as given in Theorem 3.1, and then, for a given number of processors in the optimal schedule, a task set which can achieve the worst-case upper bound under ***Algorithm RMNF-WC*** is constructed. The later is given in Theorem 3.2.

Theorem 3.1: For all sets of tasks, $N \leq (2/(ln2))N_0 + 1 \approx 2.88N_0 + 1$, 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-WC***.

Proof: For a processor j, let $\tau_1, \tau_2, \ldots, \tau_s$ be the tasks that have already assigned to processor j, and τ_{s+1} be the first task assigned to processor $j+1$. According to ***Condition WC***, we have

$$\sum_{k=1}^{s} u_k + u_{s+1} > ln2. \qquad \textbf{(E.Q.1)}$$

Let $U_j = \sum_{k=1}^{s} u_k$, for $1 \leq j \leq N$.

Since $U_{j+1} \geq u_{s+1}$, $U_j + U_{j+1} > ln2$ from (E.Q.1), where $1 \leq j \leq N - 1$.

Summing up the $N - 1$ equations yields $2\sum_{j=1}^{N} U_j - U_1 - U_N > (N - 1)\, ln2$. In other words, $2\sum_{j=1}^{N} U_j > (N - 1)\, ln2$.

Since $N_0 \geq \sum_{j=1}^{N} U_j$, $N \leq (2/(ln2))N_0 + 1$. Q.E.D.

Theorem 3.2: Let N be the number of processors required to feasibly schedule a set of tasks by ***Algorithm RMNF-WC***, and N_0 the minimum number of processors required to feasibly schedule the same set of tasks. Then $\lim_{N_0 \to \infty} N/N_0 \geq 2.87$. Together with ***Theorem 3.1***, it is concluded that $\Re(RMNF) = 2/(\ln 2)$.

Proof: Let K be a positive integer divisible by 7, i.e., $K = 7*m$, where m is a natural number, and let δ be a very small positive number and $\delta = n\varepsilon$, where n is a very large positive integer and ε is a very small positive number. The relationship between n and ε is given as follows: Given any small number δ, n is chosen large enough and ε small enough such that $\ln 2 + n\varepsilon \geq n(2^{1/n} - 1)$ and $\delta = n\varepsilon$.

The set of tasks consists of two set of groups of tasks, with the numbers of groups equal to $20K/7$ in the first set, and $\lfloor ((14K)/7)/20 \rfloor$ in the second set, where $\alpha = 1 - 5(\ln 2 - 1/2) =$ 0.034264. In terms of m, the numbers of task groups are equal to $20m$ in the first set, and $\lfloor (2m)/20 \rfloor$ in the second set. In the first set of groups of tasks, it consists of $10m$ pairs of task groups, each of which has $(n + 1)$ tasks. Note that in the (x, y) notation, x and y denote the computation time and the period of a task, respectively. A pair of task groups is given by

$$(\ln 2 - 1/2, 1), \underbrace{(\varepsilon, 1), \ldots\ldots, (\varepsilon, 1)}_{n},$$

$$(1/2, 1), \underbrace{(\varepsilon, 1), \ldots\ldots, (\varepsilon, 1)}_{n}.$$

In the second set of groups, it has $\lfloor (2m)/20 \rfloor$ groups, each of which has 20 tasks, as given by

$$\underbrace{(\alpha - 10\delta, 1), \ldots\ldots, (\alpha - 10\delta, 1)}_{20},$$

In the ***RMNF-WC*** schedule, the first set of task groups uses $20m$ processors, since $\ln 2 - 1/2 + n\varepsilon + 1/2 > n(2^{1/n} - 1)$, as illustrated in Figure 1. The second set of task groups uses $\lfloor (2m)/20 \rfloor$ processors in total, since $20(\alpha - 10\delta) + (\alpha - 10\delta) \approx 0.719 - 210\delta > 20 * (2^{1/21} - 1) \approx 0.705$, for small δ.

In the optimal schedule, the $10m$ tasks with utilization factor of $(1/2, 1)$ can be scheduled using $5m$ processors. The $10m$ tasks with utilization factor of $(\ln 2 - 1/2, 1)$ and the $20mn$ tasks with utilization factor of ε can be scheduled on $2m$ processors, with a total utilization of $2m(\alpha - 10\delta)$ left unused. This amount of utilization, i.e., $2m(\alpha - 10\delta)$, is used to execute the task groups in the second set, since $\lfloor (2m)/20 \rfloor * (\alpha - 10\delta) * 20 < 2m(\alpha - 10\delta)$.

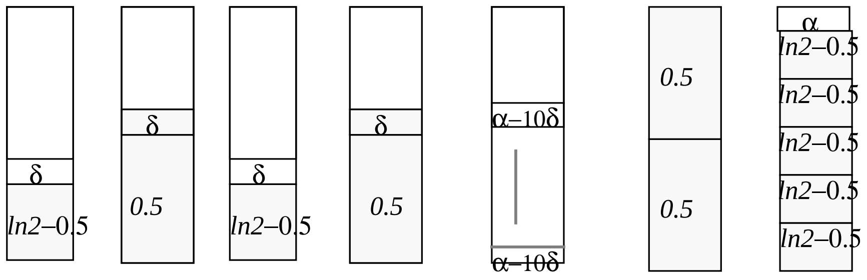
Therefore, the total number of processors used in the optimal schedule is $N_0 = 5m + 2m = 7m$, while the total number of processors used in the ***RMNF*** schedule is $N = 20m + \lfloor (2m)/20 \rfloor$. The performance bound is thus given by

$$\lim_{m \to \infty} \frac{N}{N_0} = \frac{20m + \lfloor 2m/20 \rfloor}{7m} \geq 2.87.$$

Since $N \leq (2/(\ln 2))N_0 + 1$ from ***Theorem 3.1***, it is concluded that

$$\Re = \lim_{m \to \infty} \frac{N}{N_0} = 2/(\ln 2). \text{ Q.E.D.}$$

Note that the number of processors required to execute the same task sets given in Theorem 3.2 will not be the same if the schedulability condition used is ***Condition IP***. On all processors each with a utilization equal to $u = ln2 - 0.5 + \delta$, $2e^{-u} - 1 \approx 0.648$, which implies that those tasks each with a utilization of 0.5 would not have been assigned to the next processor had ***Condition IP*** been used.



(a) ***RMNF-WC*** Schedule (b) ***Optimal*** Schedule

Figure 1: ***RMNF-WC*** **vs** ***Optimal***

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

In assigning tasks to processors, ***Algorithm RMNF-WC*** 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-WC Algorithm*** always starts to check the schedulability of a task on processors with lower indexes, i.e., those processors where some tasks have been assigned. This algorithm is given as follows:

Algorithm RMFF-WC: 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 $\tau_1, \tau_2, \ldots, \tau_n$ will be scheduled in that order. To schedule τ_i, find the least j such that task τ_i, together with all the tasks that have been assigned to processor P_j, can be feasibly scheduled according to ***Condition WC*** for a single processor, and assign task τ_i to P_j.

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

Algorithm RMFF-WC (Input: task set Σ; Output: m)

1. Set $i = 1$ and $m = 1$. /* i denotes the ith task, m the number of processors allocated*/
2. (a) Set $j = 1$. /* j denotes the jth processor */

   (b) If $U_j + u_i \leq (k_j + 1)(2^{1/(k_j+1)} - 1)$, assign task τ_i to P_j, set $k_j = k_j + 1$ 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 τ_i. Otherwise, increment $j = j + 1$ and go to step 2(b).

3. If $i > n$, i.e., all tasks have been assigned, then return m. Otherwise increment $i = i + 1$ and go to step 2(a).

When the algorithm returns, the value in m is the number of processors required to execute a given set of tasks. 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-WC*** would have better performance than that of ***Algorithm RMNF-WC***, which is indeed the case as shown by Theorem 4.1. Before proving the upper bound, however, a number of lemmas need to be established.

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

Proof: The proof is by induction. u_i is the utilization of task i, for $1 \le i \le m$.

(1) $m = 2$, $u_1 + u_2 > 2(2^{1/2}-1) = m(2^{1/2}-1)$. Therefore, the lemma is true.

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

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

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

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

Multiplying k -1 on both sides of equation (E.Q.2) yields

$$(k-1)\sum_{i=1}^{k} u_i > (k-1)\,k(2^{1/2}-1) \qquad \textbf{\textit{(E.Q.4)}}$$

Adding up equations (E.Q.3) and (E.Q.4) 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.

Lemma 4.2: If tasks are assigned to the processors according to the ***RMFF-WC 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 a utilization less than or equal to $(2^{1/2}-1)$. Let τ_j be the task a with utilization equal to u_j, that is assigned to processor P_j, and τ_k be the task with a utilization equal to u_k, that is assigned to processor P_k, with $j < k$, such that

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

Summing up these two inequalities yields

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

This implies that tasks τ_j and τ_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-WC 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 or equal to $2(2^{1/3}-1)$.

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

$$u_{j,1} + u_{j,2} \le 2(2^{1/3}-1) \text{ and}$$
$$u_{k,1} + u_{k,2} \le 2(2^{1/3}-1), \qquad \textbf{(E.Q.5)}$$

where $u_{x,l}$ denotes the utilization of task $\tau_{x,l}$. There are three cases to consider.

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-WC***, we must have

$$u_{j,1} + u_{j,2} + u_{k,1} > 3(2^{1/3}-1) \text{ and}$$
$$u_{j,1} + u_{j,2} + u_{k,2} > 3(2^{1/3}-1).$$

Summing up these two inequalities, we have

$$u_{k,1} + u_{k,2} > 6(2^{1/3}-1) - 2(u_{j,1} + u_{j,2}) > 2(2^{1/3}-1)$$

which is a contradiction to (E.Q.5).

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-WC***, we must have

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

Summing up these two inequalities, we have

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

which is again a contradiction to (E.Q.5).

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-WC***, we must have

$$u_{j,1} + u_{k,1} > 2(2^{1/2}-1) \text{ and}$$
$$u_{j,1} + u_{j,2} + u_{k,2} > 3(2^{1/3}-1).$$

Summing up these two inequalities, we have

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

which is again a contradiction to (E.Q.5). 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 4.4: If tasks are assigned to the processors according to the ***RMFF-WC Algorithm***, among all processors to each of which n tasks are assigned, there is at most one processor for which the utilization factor of the set of the n tasks is less than or equal to $n(2^{1/(n+1)}-1)$. $\lim_{n \to \infty} n(2^{1/(n+1)} - 1) = ln2$

Theorem 4.1: Let N be the number of processors required to feasibly schedule a set of tasks by the ***Algorithm RMFF-WC***, 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 utilization 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

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

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

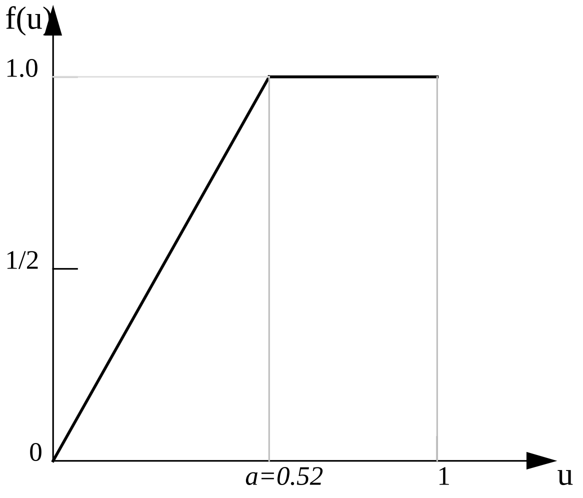


Figure 2: **Mapping Function for *RMFF-WC* and *RMBF-WC***

deficiency δ_j of processor P_j is defined as

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

The coarseness α_j of processor P_j is defined as

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

Lemma 4.5: For ***Algorithm RMFF-WC***, 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 α, then the utilization of each task that was assigned to P exceeds α.

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

Lemma 4.6: 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 $u_2 < a$, since $a \approx 0.52$. $\sum_{i=1}^{m} f(u_i) = f(u_1) + \sum_{i=2}^{m} f(u_i) = 1 + (\sum_{i=2}^{m} u_i) / a \le 1 + (1 - a) / a = 1 / a$. 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 4.7: Suppose tasks are assigned to processors according to ***RMFF-WC 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 it.

Proof: According to Lemma 4.5, $u_i > \alpha \ge a / 3$ 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 4.8: Suppose tasks are assigned to processors according to ***RMFF-WC 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 4.9: Suppose tasks are assigned to processors according to ***RMFF-WC Algorithm.*** If a processor with coarseness α 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) $m \ge 3$ and $\sum_{i=1}^{m} u_i \le ln2 - \alpha - a\beta$.

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

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

(3) If properties (1) and (2) do not hold, then $m \ge 3$. Since $\sum_{i=1}^{m} f(u_i) < 1$, α must be less than $a / 3$ and $\sum_{i=1}^{m} u_i < ln2 - \alpha$ according to Lemma 4.7 and Lemma 4.8. Let $\sum_{i=1}^{m} u_i = ln2 - \alpha - \gamma$, where $\gamma > 0$. To find out the relationship between γ and β, let us replace the first three tasks τ_1, τ_2, and τ_3 by three new tasks with utilizations υ_1, υ_2, and υ_3, such that $\upsilon_1 + \upsilon_2 + \upsilon_3 = u_1 + u_2 + u_3 + \gamma$, $\upsilon_1 \ge u_1$, $\upsilon_2 \ge u_2$, $\upsilon_3 \ge u_3$, and $\upsilon_1 < a$, $\upsilon_2 < a$, $\upsilon_3 < a$. According to Lemma 4.8, $f(\upsilon_1) + f(\upsilon_2) + f(\upsilon_3) + \sum_{i=4}^{m} f(u_i) \ge 1$. Since $f(\upsilon_1) + f(\upsilon_2) + f(\upsilon_3) = f(u_1) + f(u_2) + f(u_3) + f(\gamma) = f(u_1) + f(u_2) + f(u_3) + \gamma / a$, $\gamma / a + 1 - \beta \ge 1$. $\gamma \ge a\beta$. Therefore, $\sum_{i=1}^{m} u_i \le ln2 - \alpha - a\beta$. 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 4.6, $\varpi \le N_0 / a$, where $a =$

$2(2^{1/3}-1)$.

Suppose that among the N processors that are used by ***RMFF-WC Algorithm*** to schedule a given set Σ 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 4.3, there is at most one processor whose utilization in the ***RMFF-WC*** 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 4.2, 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 α_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 $1 \leq i \leq n_3$. For processor Q_i, $U_i \leq ln2 - \alpha_i - a\beta_i$ according to Lemma 4.9.

According to the definition of coarseness, $\alpha_{i+1} \geq \delta_i \geq ln2 - U_i$. Therefore $\alpha_{i+1} \geq \alpha_i + a\beta_i$, for $1 \leq i < n_3$. Summing up these $(n_3 - 1)$ equations yields

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

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

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

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

Since $\varpi \leq N_0 / a$ by Lemma 4.6,

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

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

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

Having proven the upper bound, we are now ready to construct a number of task sets which indeed require the upper-bounded number of processors according to ***RMFF-WC Algorithm.***

Theorem 4.2: Let N be the number of processors required to feasibly schedule a set of tasks by ***RMFF-WC 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: In order to find the bound $\Re = \lim_{N_0 \to \infty} N/N_0$, we proceed by finding the maximum number of processors needed to schedule a certain set of tasks using ***RMFF-WC 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-WC Algorithm***, in other words, they appear last in the task set.

Region 1: There are $2N_0$ number of tasks, each of which has a utilization of $u_1 = (2^{1/2} - 1) + \varepsilon$, where ε is a arbitrary small number. These $2N_0$ tasks will utilize $2N_0$ number of processors in the ***RMFF-WC*** schedule, while requires only N_0 number of processors in the processors in the optimal schedule. If $N_0 \leq 2$, then we have found $\Re = 2$.

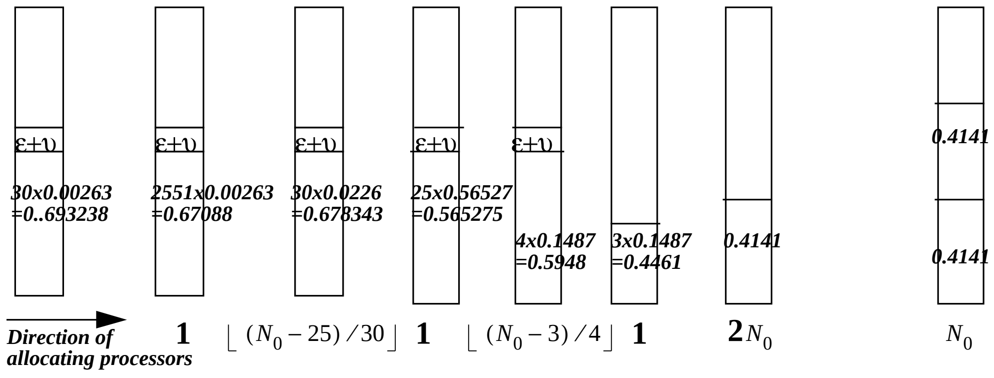
Region 2: If $3 \leq N_0 \leq 5$, there are N_0 tasks, each of which has a utilization of $u_2 = (2^{1/5} - 1)$. These N_0 tasks utilize one processors in the ***RMFF-WC*** schedule, while requires no extra processor in the optimal schedule, only to fill part of the utilization left by tasks in region 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 $u_1 + 3u_2 > 4(2^{1/4} - 1)$. $N = 2N_0 + 1$. The bound is given by $\Re = 2N_0 / N_0 + 1 / N_0$.

Region 3: If $6 \leq N_0 \leq 9$, 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 having 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-WC*** 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 $10(2^{1/10} - 1) - (3u_2 + 6u_3) < u_2$, 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 $10 \leq N_0 \leq 12$, 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) + ε, where ε is an arbitrary small number. These four tasks are placed in one processor in the ***RMFF-WC*** schedule, while requires no extra processor in the optimal schedule, only to fill part of the space 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 $u_i \leq 1 - \sum_{l=1}^{i-1} u_l$, for $i \geq 2$.



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

Figure 3: ***RMFF-WC*** vs ***Optimal***

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

$$\Re = \frac{N}{N_0} = \frac{2N_0 + 1 + \lfloor (N_0 - 3)/4 \rfloor + 1 + \lfloor (N_0 - 25)/30 \rfloor + \ldots\ldots}{N_0} \approx 2.30. \qquad \textbf{\textit{(E.Q.6)}}$$

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

There are $2N_0 = 54$ number of tasks with utilization ($2^{1/2}$ - 1) + ε, where ε 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 $u_3 = 1 - 2*((2^{1/2} - 1) + \varepsilon) - (2^{1/5} - 1) - \varepsilon$. The set of tasks is given as follows. Note that the total number of tasks is 106.

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

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

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

According to ***RMFF-WC Algorithm***, The first 25 tasks are scheduled on the first processor. Since $25u_3 + u_2 = 0.571863 - 75\varepsilon + (2^{1/5} - 1) = 0.720561 > 26(2^{1/26} - 1) = 0.702469$, the 26th task is scheduled on the second processor. The 29th task can not be scheduled on the second processor, since $4u_2 + u_2 + \varepsilon = 5(2^{1/5} - 1) + \varepsilon > 5(2^{1/5} - 1)$. Proceeding in this fashion, the 23 successive tasks occupy 6 processors. The 53th task have to be scheduled on the 8th processor, since $3u_2 + u_1 = 0.446095 + (2^{1/2} - 1) + \varepsilon > 4(2^{1/4} - 1)$. The rest of the 53 tasks occupies 53 processors, one task for a processor, since $(2^{1/2} - 1) + \varepsilon + (2^{1/2} - 1) + \varepsilon > 2(2^{1/2} - 1)$. 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-WC* (and also *RMBF-WC)*

N_0	$\Re$(RMFF-WC)	N_0	$\Re$(RMFF-WC)
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.6) gives the *EXACT* tight bound for ***RMFF-WC Algorithm.*** Q.E.D.

Note that the number of processors required to execute the same task sets as given in proof of the above theorem are the same for algorithms ***RMFF*** (Liu and Layland’s) and ***RMFF-WC***. This result is seemingly counter-intuitive, since the static algorithm — ***RMFF*** takes advantages of fact that tasks are ordered according to their periods. Yet on a close inspection, we find that there is not much difference between the available utilization on a processor returned by ***Condition IP*** and that by ***Condition WC*** when that processor is quite occupied, i.e., with a reasonably high utilization. The difference is significant only when the processor is very lightly utilized. However, this difference is offset by the manner in which ***RMFF-WC*** schedules tasks.

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

When ***Algorithm RMFF-WC*** 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-WC: 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 $\tau_1, \tau_2, \ldots, \tau_n$ will be scheduled in that order. To schedule τ_i, find the least j such that task τ_i, together with all the tasks that have been assigned to processor P_j can be feasibly scheduled according to ***Condition WC*** for a single processor, and $(k_j + 1)\,(2^{1/(k_j+1)} - 1)$ - $(U_j + u_i)$ be as small as possible, and assign task τ_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 τ_i.

Surprisingly, even with this modification in assigning tasks to processors, the ***RMBF-WC Algorithm*** does not outperform ***Algorithm RMFF-WC*** in the worst-case, as shown by Theorem 5.1 and Theorem 5.2. Before we prove the tight bound for ***RMBF-WC***, 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-WC Algorithm***, they are divided into two types of processors:

Type (I): For all the tasks $\tau_{x,1}, \tau_{x,2}, \ldots, \tau_{x,m}$ with utilizations $u_{x,1}, u_{x,2}, \ldots, u_{x,m}$ that were assigned to a processor P_x in the completed ***RMBF-WC*** schedule, there exists at least one task $\tau_{x,i}$ 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 $i\,(2^{1/i} - 1)$ - $\sum_{l=1}^{i-1} u_{x,l} < (n_y + 1)\,(2^{1/(n_y+1)} - 1)$ - $\sum_{l=1}^{n_y} u_{y,l}$, 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 $\tau_{x,i}$ is, for convenience, referred to as a Type (I) task.

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

Lemma 5.1: For ***Algorithm RMBF-WC***, 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-WC***, 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-WC 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 4.3 and Lemma 4.4.

Lemma 5.3: In the completed ***RMBF-WC*** 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-WC*** 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-WC*** 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 $\tau_{k,2}$ could have been scheduled, but $2(2^{1/2}-1) - u_{k,1} < (n_y+1)(2^{1/(n_y+1)}-1) - \sum_{l=1}^{n_y} u_{y,l}$, 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 $u_{k,1} > (n_y+1)(2^{1/(n_y+1)}-1) - \sum_{l=1}^{n_y} u_{y,l}$ (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), $u_{k,1} > (n_y+1)(2^{1/(n_y+1)}-1) - \sum_{l=1}^{n_y} u_{y,l} > 2(2^{1/2}-1) - u_{k,1}$. Therefore $u_{k,1} > (2^{1/2}-1)$. Q.E.D.

Lemma 5.6: In the completed ***RMBF-WC*** 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 tight bound for ***RMBF-WC Algorithm.***

Lemma 5.7: In the completed ***RMBF-WC*** 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

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

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 $(n_y + 1)(2^{1/(n_y+1)} - 1) - \sum_{x=1}^{n_y} u_{i,x} \leq 3\,(2^{1/3}-1) - (u_{i,1} + u_{i,2}) < 3\,(2^{1/3}-1) - u_{i,1}$,

$2(2^{1/2}-1) - u_{j,1} < 3\,(2^{1/3}-1) - u_{i,1}$.

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

$u_{i,1} < u_{j,1}$. ***(E.Q.7)***

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

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

Since $2(2^{1/2}-1) - u_{i,1} < (n_z + 1)(2^{1/(n_z+1)} - 1) - \sum_{x=1}^{n_z} u_{j,x} < 2(2^{1/2}-1) - u_{j,1}$, $u_{i,1} > u_{j,1}$. This is a contradiction to equation (E.Q.7).

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

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

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 $(n_y + 1)(2^{1/(n_y+1)} - 1) - \sum_{x=1}^{n_y} u_{i,x} \leq 3\,(2^{1/3}-1) - (u_{i,1} + u_{i,2}) < 3\,(2^{1/3}-1) - u_{i,1}$,

$2(2^{1/2}-1) - u_{j,1} < 3\,(2^{1/3}-1) - u_{i,1}$.

Therefore for processors P_i and P_j, we have

$2(2^{1/2}-1) - u_{j,1} < 3(2^{1/3}-1) - u_{i,1}$. **(E.Q.8)**

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

$2(2^{1/2}-1) - u_{k,1} < 3(2^{1/3}-1) - u_{j,1}$ **(E.Q.9)**

$2(2^{1/2}-1) - u_{l,1} < 3(2^{1/3}-1) - u_{k,1}$ **(E.Q.10)**

Summing up equations (E.Q.8), (E.Q.9), and (E.Q.10) yields $u_{l,1} > 3(2(2^{1/3}-1) -3(2^{1/3}-1)) + u_{i,1}$. Since $u_{i,1} > (2^{1/2}-1)$ according to Lemma 5.5, $u_{l,1} > 0.5342 > 2(2^{1/3}-1)$. This results in a contradiction to $\sum_{x=1}^{n_l} u_{l,x} < 2(2^{1/3}-1)$. Q.E.D.

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

For a processor P_j, its deficiency δ_j and its coarseness α_j are similarly defined as those for ***RMFF-WC Algorithm***. Also note that Lemma 4.7, Lemma 4.8, and Lemma 4.9 also hold for those processors of Type (II) in the ***RMBF-WC*** 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 $\sum_{i=1}^{m} f(u_i) \le 1/a$, 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, $\varpi \le N_0 / a$, where $a = 2(2^{1/3} - 1)$.

Suppose that among the N processors that are used by ***RMBF-WC Algorithm*** to schedule a given set Σ of tasks, M_1 of them belong to the 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 $m \ge 2$ such that the mth task is a Type (I) task. For all the processors of Type (I) on each of which the mth task is a Type (I) task with $m \ge 3$, $\sum_j f(u_j) > 1$ 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-WC*** 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-WC*** schedule

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

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

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-WC 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.

Table 2: Performance of Several Multiprocessor Scheduling Algorithms

	Condition WC	***Condition IP***	***Condition IFF***
Next-Fit	2.88	2.67	2.88
First-Fit	2.33	2.33 [11]	?
Best-Fit	2.33	2.33 [11]	?
FFDUF	2.0	?	?

VI. Concluding Remarks

In this paper, we investigate the problem of scheduling a set of periodic tasks on a multiprocessor system so as to minimize the number of processors used. Three scheduling algorithms — ***RMNF-WC***, ***RMFF-WC***, and ***RMBF-WC***, which use ***Condition WC*** as schedulability condition, are proposed, and their worst-case performance investigated. Since ***Condition WC*** does not require any apriori knowledge about an incoming task, the three algorithms are dynamic algorithms. Surprisingly, except for ***RMNF-WC***, the dynamic algorithms have the same worst-case performance bounds as their static counterparts using ***Condition IP***. As a summary, the performance of several scheduling heuristics is presented in Table 2, where ? represents an open problem.

Our future work will focus on the investigation of the scheduling heuristics under the necessary and sufficient condition -- ***Condition IFF***. Even though we have proven (not presented here) that the performance of Rate-Monotonic-Next-Fit does no better under ***Condition IFF*** than ***Condition WC***, we have reasons to believe that Rate-Monotonic-First-Fit and Rate-Monotonic-Best-Fit will perform better under ***Condition IFF*** than ***Condition WC*** or ***Condition IP***.

Appendix

Lemma 4.4: If tasks are assigned to the processors according to the ***RMFF-WC Algorithm***, among all processors to each of which n 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 4.2 and Lemma 4.3. 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

$$\sum_{m=1}^{k+1} u_{i,m} < (k + 1)(2^{1/(k+2)}-1) < k(2^{1/(k+1)} - 1) \qquad \textbf{\textit{(E.Q.11)}}$$

and $\sum_{m=1}^{k+1} u_{j,m} < (k + 1)(2^{1/(k+2)}-1) < k(2^{1/(k+1)} - 1)$. ***(E.Q.12)***

Since processors P_i and P_j are each assigned $n = k + 1$ tasks, we must have
$\sum_{m=1}^{k+1} u_{i,m} \le (k + 1)(2^{1/(k+1)}-1)$ and
$\sum_{m=1}^{k+1} u_{j,m} \le (k + 1)(2^{1/(k+1)}-1)$

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 Pi, $1 \le x \le k+1$. 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: $1 \le x \le k+1$. Since $\Delta^i + u_{j,z} > (k + 2)(2^{1/(k+2)}-1)$,
$u_{j,z} > (k + 2)(2^{1/(k+2)}-1) - \Delta^i > (k + 2)(2^{1/(k+2)}-1) - (k + 1)(2^{1/(k+2)}-1)$
$= 2^{1/(k+2)} - 1$, for $x \le z \le k+1$, and
$\Delta^i - u_{i,k+1} + u_{j,z} > (k + 1)(2^{1/(k+1)}-1)$, for $1 \le z < x$.
$u_{j,z} > (k + 1)(2^{1/(k+1)}-1) - \Delta^i + u_{i,k+1} > (k + 1)(2^{1/(k+1)}-1) - (k + 1)(2^{1/(k+2)}-1)$
$= (k+1)(2^{1/(k+1)} - 2^{1/(k+2)}) > 2^{1/(k+2)} -1$
$\Delta^j = \sum_{m=1}^{x-1} u_{j,m} + \sum_{m=x}^{k+1} u_{j,m} > (k+1)(2^{1/(k+2)}-1)$,
which is a contradiction to equation (E.Q.12).

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.
$\Delta^i - u_{i,k+1} + u_{j,z} > (k + 1)(2^{1/(k+1)}-1)$, for $1 \le z \le k+1$.
$u_{j,z} > (k + 1)(2^{1/(k+1)}-1) - \Delta^i + u_{i,k+1} > (k + 1)(2^{1/(k+1)}-1) - (k + 1)(2^{1/(k+2)}-1)$
$= (k+1)(2^{1/(k+1)} - 2^{1/(k+2)}) > 2^{1/(k+2)} -1$
$\Delta^j = \sum_{m=1}^{k+1} u_{j,m} > (k+1)(2^{1/(k+2)}-1)$,
which is a contradiction to equation (E.Q.12). Q.E.D.

Lemma 5.6: In the completed ***RMBF-WC*** 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 τ_m could have been scheduled, but $m(2^{1/m}-1) - \sum_{j=1}^{m-1} u_{k,j} < (n_y + 1)(2^{1/(n_y+1)} - 1) - \sum_{l=1}^{n_y} u_{y,l}$, 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 $u_{k,i} > (n_y + 1)(2^{1/(n_y+1)} - 1) - \sum_{l=1}^{n_y} u_{y,l}$ (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 $1 \le i \le m-1$, $u_{k,i} > (n_y + 1)(2^{1/(n_y+1)} - 1) - \sum_{l=1}^{n_y} u_{y,l} > m(2^{1/m}-1) - \sum_{j=1}^{m-1} u_{k,j}$. Summing up these $(m-1)$ inequalities yields

$\sum_{j=1}^{m-1} u_{k,j} > (m-1)m(2^{1/m}-1) - (m-1)\sum_{j=1}^{m-1} u_{k,j}$. Therefore, $\sum_{j=1}^{m-1} u_{k,j} > (m-1)(2^{1/m}-1)$. 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.Y.T. LEUNG AND J. WHITEHEAD. “On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks,” *Performance Evaluation* **2**, 237-250 (1982).

[10] 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)

[11] Y. OH AND S.H. SON, “Tight Bounds of Heuristics for a Real-Time Scheduling Problem,” *Submitted for Publication*, April 1993.

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