Fixed-Priority Scheduling of Periodic Tasks on Multiprocessor Systems

Yingfeng Oh and Sang H. Son

Department of Computer Science, University of Virginia

Thornton Hall, Charlottesville, VA 22903, USA

E-mail: {yo5u, son}@virginia.edu

Abstract

Consider the problem of periodic task scheduling, in which we seek to minimize the total number of processors required to execute a set of tasks such that task deadlines are guaranteed by the *Rate-Monotonic* (or RM) algorithm on each processor. This problem was first investigated by Dhall and Liu, and the previous lowest bound for the problem was 2.0. In this paper, an improved solution is given by designing a new algorithm for it. The algorithm, called *RM-First-Fit-Decreasing-Utilization* (or RM-FFDU), is shown to have a worst-case tight bound of $5/3 = 1.66\ldots$, the lowest upper bound ever derived for the scheduling problem. Simulation studies show that on the average, the new algorithm performs consistently better than those in the literature.

1 Introduction

In this paper, we study the the problem of scheduling a set of periodic tasks on a multiprocessor system such that task deadlines are guaranteed on individual processors by the rate-monotonic scheduling algorithm. The problem is stated as follows: *Given a set of n periodic tasks $\Sigma = \{\tau_i = (C_i, T_i) | i = 1, 2, \ldots, n\}$, where each task τ_i is characterized by its computation time C_i and its period T_i, what is the minimum number of processors required to execute the tasks such that their deadlines are met by the rate-monotonic algorithm?*

The scheduling problem, referred to as the *Rate-Monotonic Multiprocessor Scheduling* (or RMMS), is a natural generalization of the uniprocessor periodic task scheduling problem studied in [18]. Since this problem has been proven to NP-complete [16], we shall focus on fast heuristic algorithms for solving it, seeking to prove close bounds on the extent to which they can deviate from optimality. Due to the complexity involved, the analysis of simple approximation methods for the problem represents a constant challenge.

It is a common practice to analyze the performance ratio of the algorithm under study when working with approximation algorithms for combinatorial optimization problems [10]. Let $A(I)$ denote the performance of a given algorithm for an instance I of a particular combinatorial optimization problem and let $OPT(I)$ denote the performance of an optimal algorithm for the same instance. The ratio of $A(I)$ to $OPT(I)$, considered over all instances I, provides us with an indicator of the quality of the given algorithm. To be specific for the scheduling heuristics for RMMS, let $N_A(\Sigma)$ (or N_A) and $N_0(\Sigma)$ (or N_0) denote the number of processors required by the heuristic A and the optimal number of processors required to schedule a given set Σ of tasks, respectively. An optimal algorithm for the scheduling problem is the one that uses the minimum number of processors to schedule any task set. Then, the worst-case bound for heuristic A is determined by

$$\Re_A = \inf\{r \geq 1: \frac{N_A(L)}{N_0} \leq r \text{ for all lists } L\},$$

It is apparent that $\Re_A$ always assumes a value no less than one, and the smaller the

$\Re_A$'s value is, the better the heuristic algorithm A performs in terms of the worst-case scenario. In other words, the smaller the $\Re_A$'s value is, the closer the heuristic solution is to the optimal one. Hence, we seek to minimize $\Re_A$ as much as possible in designing a heuristic algorithm.

Despite the fact that the rate-monotonic algorithm has become very populous among practitioners in solving many practical problems, and among researchers in generalizing the original results (see, e.g., [21]), previous results on the RMMS problem have been limited and the lack of progress in the area has been notable. The best bound was 2.00, which was derived about a decade ago, when the use of multiprocessor was not very common. Now the employment of multiprocessors for many real-time applications is a necessity.

These factors have led some researchers to believe that using RM scheduling algorithm in a multiprocessor environment is not very efficient.

In this paper, we address the problem by designing a new heuristic called *RM-First-Fit-Decreasing-Utilization* (or RM-FFDU) for it. While the design of such heuristic algorithms can be straightforward, the analysis of their performance presents a major intellectual challenge. By employing such techniques as weighting functions, we prove that RM-FFDU has a worst-case tight bound of 5/3. This bound reflects a significant improvement over the previous ones, and it shows that, contrary to the pessimistic belief held by some researchers about the efficiency of heuristics for the RMMS problem, it is possible to develop very provably good heuristics for it.

The superiority of the new algorithm can been readily seen by comparison with the existing heuristic algorithms from the literature in Table 1. The measure $O(nlogn)$ denotes the computation time complexity for scheduling a set of n tasks. Dhall and Liu proposed two heuristic algorithms for the scheduling problem: *Rate-Monotonic-Next-Fit* (or RMNF) and *Rate-Monotonic-First-Fit* (or RMFF) [8]. It was shown that $2.4 \leq \Re_{RMNF} \leq 2.67$, and $2 \leq \Re_{RMFF} \leq 4 \times 2^{1/3}/(1+2^{1/3}) \approx 2.23$. Unfortunately, the upper bound derived for RMFF was incorrect due to some errors in their proof [19]. Davari and Dhall later considered two other algorithms called *First-Fit-Decreasing-Utilization-Factor* (or FFDUF) and *NEXT-FIT-M* (or NF-M) [5, 6]. Their worst-

Existing Algorithm					New Algorithm
Scheme A	RMNF	RMFF	NF-M	FFDUF	RM-FFDU
$\Re_A$	2.67	2.23?	2.28	2.0	1.66...
Complexity	$O(n)$	$O(n \log n)$	$O(n)$	$O(n \log n)$	$O(n \log n)$

Table 1: Comparison of Heuristic Scheduling Algorithms

case performance bounds are $\Re_{FFDUF} \leq 2$ and $\Re_{NF-M} \leq S_M$, respectively, where $S_M = 2.34$ for $M = 4$, and $S_M \rightarrow 2.2837$ for $M \rightarrow \infty$. Most recently, Burchard, Liebeherr, Oh, and Son proposed an algorithm called RMGT, that has a worst-case bound of 1.75 [3].

Our approach to tackling the problem is to use a recently derived schedulability condition that exhibits good performance while remaining simple enough so that the worst-case performance analysis is still possible. We apply the familiar bin-packing heuristic – First-Fit-Decreasing [11] to allocate tasks to processors. In the analysis of the worst-case performance, we not only obtain the upper bound of the algorithm, but also provide examples that show that the bound is tight. In the light of providing the worst-case performance, our analysis is intricate, because our algorithms are more complicated than their bin-packing counterparts, in the sense that the size of a bin is unitary, while the "size" or utilization of a processor is a variable. The value of the variable is determined by a function that is called the schedulability condition.

The solution to such a scheduling problem has practical implications. On the one hand, the real-time application domain is becoming increasingly large. As requirements of real-time support for industrial applications become more sophisticated, the employment of multiprocessors to meet computational power requirement seems inevitable. On the other hand, the state-of-the-art of hardware technology makes multiprocessor support a reality. The scheduling of periodic tasks on a multiprocessor thus becomes an urgent problem that needs to be solved. Multiprocessor heuristic algorithms based on RM scheduling have many advantages that are inherent in RM scheduling. For example, RM scheduling ensures that as long as the CPU utilization of all tasks lies

below a certain bound, all tasks will meet their deadlines without the programmer knowing exactly when any given instance of a task is running. Even if a transient overload occurs, a fixed subset of critical tasks will still meet their deadlines as long as their total CPU utilization lies within certain bound. The rate-monotonic scheduling discipline has been widely used in a number of applications. For example, it has been specified for use with Space Station on-board software as the means for scheduling multiple independent task execution; it will be built into the on-board operating system [9]. Many Ada compilers also support this scheduling discipline [21]. More recently, this scheduling discipline is used to schedule video, audio, and data streams in a multimedia system.

The rest of the paper is organized as follows: In Section 2, we present the task model upon which the scheduling problem is defined. In Section 3, the new heuristic algorithm RM-FFDU is designed. The major portion of this paper is the proof of the worst-case performance bound of RM-FFDU, which appears in Section 4. In Section 5, we present our simulation methodology and the results of the experiments. We conclude in Section 6 with a look at future research directions.

2 A Task Model for Real-Time Systems

We assume that processors are homogeneous or identical, in the sense that the run-time of a task remains the same across all processors. For a set of n tasks, we assume:

(1) The requests of each task are periodic, with constant interval between requests.

(2) The deadline constraints specify that each request must be completed before the next request of the same task occurs.

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

(4) The worst-case run-time (or computation time) for the request of a task is known for the task. Run-time here refers to the time a processor takes to execute the request without interruption.

For a task τ with a period of T, the request of task τ comprises the readiness of

task τ for execution and its CPU request. If the initial release time of task τ is R, then the task will arrive in the system at time $R + kT$ and we say that there is a request from task τ every T time units for execution, where $k \in Z^+$ and Z^+ is the set of natural numbers. According to assumption (2), a request from task τ at time instance $(R + kT)$ must be completed before the next request of the same task arrives, i.e., before time instance $[R + (k + 1)T]$, where $k \in Z^+$. A task meets its deadline if all its requests meet their deadlines.

We say that a set of tasks is feasible if it can be scheduled by some algorithms such that all task deadlines are met. If a set of periodic tasks can be feasibly scheduled on a single processor using fixed-priority scheduling, then the *Rate-Monotonic* [18] or *Intelligent Fixed Priority algorithm* [20] is optimal, in the sense that if a set of periodic tasks is feasible with a fixed-priority algorithm, then it is feasible with the rate-monotonic algorithm. In fixed-priority scheduling, the priority of a task remains fixed once it is assigned. It was also shown that for the fixed-priority scheduling, the task release times do not affect the schedulability of the tasks and hence a task is completely defined by two numbers: its computation time and its request period. 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. In other words, a task with a shorter period is assigned a higher priority. The execution of a low-priority task will be preempted if a high-priority task arrives. Since each task has a potentially infinite number of requests, it would be rather time-consuming to manually check that each request finishes before its corresponding deadline. Fortunately, Liu, Layland, and others have provided us with simple schedulability conditions that can be used for that purpose [18, 20, 7, 19]. Because the rate-monotonic algorithm is the best fixed-priority algorithm in the same sense as its optimality, and its ease of implementation due to the fixed-priority manner, we will use it in guaranteeing task deadlines on each processor.

Although the task model for the rate-monotonic scheduling discipline is only a simplified one with regard to most practical real-time applications, researchers through the years have successfully developed a host of scheduling techniques out of this discipline to solve many practical real-time problems, such as task synchronization, bus

scheduling, joint scheduling of periodic and aperiodic tasks, transient overload, and parallel processing (see, e.g., [22, 21]). The scheduling discipline has proved to be far more powerful than it was expected.

While rate-monotonic scheduling is optimal for a uniprocessor system, it is, unfortunately, not so for a multiprocessor system. In fact, the problem of optimally scheduling a set of periodic tasks on a multiprocessor system using either fixed-priority or dynamic priority assignment is known to be NP-complete [16]. Hence, any practical solution to the RMMS problem presents a trade-off between computational complexity and performance. It has been shown that heuristic algorithms can deliver near-optimal solutions to NP-complete problems with limited computational overhead [10]. Therefore, we seek practical, efficient approximation algorithms for the RMMS problem in hopes of guaranteeing near-optimal results.

For the convenience of presentation, we adopt the following notations: processors are numbered in the order consistent with that of allocating them. We shall denote a task τ_i by the ordered pair (C_i, T_i), where C_i and T_i are the computation time and the period of the requests, respectively. The ratio C_i/T_i, which is denoted as u_i, is called the utilization (or load) of the task τ_i. $\tau_{x,l}$ 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. U_j denotes the total CPU utilization (or load) of the jth processor. $\tau = (x, y)$ characterizes a task τ, where x and y are the computation time and the period of task τ, respectively.

3 The Design of RM-FFDU

The general solution to such a problem as RMMS involves two algorithms: one to assign tasks to individual processors, and the other to schedule tasks assigned on each individual processor. Two major schemes exist for assigning tasks to processors: partitioning and non-partitioning schemes. In a non-partitioning scheme, each occurrence of a task may be executed on a different processor, while a partitioning scheme requires

that all occurrences of a task are executed on the same processor. The partitioning scheme is often preferred, because relatively low overhead is involved in the scheduling process. A scheduling algorithm can also be classified as a dynamic (on-line) or static (off-line) algorithm according to whether the algorithm requires *a priori* knowledge about the whole task set or not. If an algorithm requires that the entire task set be known, then the algorithm is referred to as being static, otherwise, it is said to be dynamic.

To solve the RMMS problem, we need to address these two issues: the scheduling on each processor and the assignment of tasks to processors. In general, the performance of a heuristic algorithm to solve such problem depends on the strategies taken to solve each of the two issues. One simple solution to the problem will be to use one processor for the execution of one task. Although this solution guarantees that each request of a task meets its deadline, it is very inefficient in processor usage. Our goal is to use as few processors as possible to accommodate a given set of tasks, as stated in the RMMS problem. In the following, we will first present the schedulability condition that will be used in RM-FFDU and then the strategy to assign tasks to processors.

For a set of n tasks $\Sigma = \{\tau_i = (C_i, T_i) | i = 1, 2, \ldots, n\}$ to be scheduled on a single processor, Liu and Layland provides us with a schedulability condition that if $\sum_{i=1}^{n} C_i/T_i \leq n(2^{1/n} - 1)$, then all the n tasks can be scheduled to meet their deadlines by the rate-monotonic algorithm. We would like to use this condition in determining the feasibility of a set of tasks, but the performance of the heuristics using this condition in solving the RMMS problem is not as good, as shown by the previous work [8, 5, 6]. This is because the condition is a worst-case one; there are some task sets that are feasible with the rate-monotonic algorithm, but cannot be determined to be feasible by the condition.

A necessary and sufficient condition has been found for the rate-monotonic algorithm [13, 15]. It is clear that the upper bound of the performance of a heuristic algorithm using the necessary and sufficient condition is no higher than that of the algorithm using any sufficient condition. However, the time complexity in using the necessary and sufficient condition for the rate-monotonic algorithm is data-dependent.

In the worst case, it may require more than exponential time complexity with respect to the number of tasks. Due to the stringent requirements of hard real-time systems, it is not practical to employ the necessary and sufficient condition in general for these systems. Hence, heuristic algorithms using simple sufficient conditions are often sought in the solution of RMMS.

In the following, we will be using a new sufficient schedulability condition for the scheduling of tasks, which is given in Theorem 1.

Theorem 1 *Let* $\Sigma = \{\tau_i = (C_i, T_i) | i = 1, 2, \ldots, n-1\}$ *be a set of* $(n-1)$ *tasks with their utilizations being* $\{u_1, u_2, \ldots, u_{n-1}\}$*, and it can be feasibly scheduled by the rate-monotonic algorithm. A new task* $\tau_n = (C_n, T_n)$ *can be feasibly scheduled together with the* $(n-1)$ *tasks on a single processor by the rate-monotonic algorithm, if*

$$C_n/T_n \leq 2[\prod_{i=1}^{n-1}(1+u_i)]^{-1} - 1 \tag{1}$$

Note that this condition subsumes the Liu and Layland's condition, and the time complexity to determine the feasibility of a set of n tasks is clearly $O(n)$. This condition is superior to Liu and Layland's condition in performance, since it always yields a processor utilization no lower than that by the Liu and Layland's condition [18]. It is also superior to the necessary and sufficient condition in time complexity, since the former runs only in time linear to the number of tasks, while the time complexity of the latter is data-dependent and is at least linear to the number of tasks. The higher processor utilization is achieved by this new condition, since it uses more information from the task set. The new condition, which is referred to as the *Utilization-Oriented* (or UO) condition, takes into account the relative values of task utilizations u_i, while Liu and Layland's condition does not consider this information. Under the worst-case situation of task utilizations u_i, the new condition degrades to the worst-case condition, i.e., $\sum_{i=1}^{n} C_i/T_i \leq n(2^{1/n} - 1)$. The proof of the theorem can be found in [19].

To assign tasks to processors, variants of well-known bin-packing heuristics can be applied, where processors and tasks are regarded as bins and items, respectively. The

bin-packing problem is concerned with packing variable-sized items into fixed-sized bins using the least number of bins (see, e.g., [4]). Among the different strategies, the First-Fit strategy has been frequently adapted to solve the various bin-packing problems and is one of the best studied ones. The First-Fit strategy is a simple, on-line one, and yet it can deliver near-optimal performance. For the classical bin-packing problem, the First-Fit heuristic has a tight bound of 1.7 [12], while no on-line algorithm can have a worst-case bound less than 1.53 [2, 17]. By ordering the items according to their decreasing sizes and applying the First-Fit strategy to pack the new list of items, we have the famous First-Fit-Decreasing (or FFD) heuristic, which is clearly off-line and has a tight bound of 11/9 [11]. (Note that there are algorithms guaranteed to produce results as close to the optimal result as desired [23, 14], but these algorithms are not practical because the time required to ensure results at most $(1+\epsilon)$ times the optimal result grows extremely fast as ϵ approaches zero.) It has been known that if the input data are preprocessed and the same heuristic strategy is employed, the performance can be improved. Davari and Dhall's FFDUF is yet an example of applying the First-Fit heuristic on a set of tasks sorted in the decreasing order of their utilizations. For these reasons, we will be using the First-Fit strategy to assign tasks to processors in the order of decreasing task utilizations. We describe the new algorithm using pseudo code as follows:

RM-FFDU (Input: task set Σ; Output: m)

(1) Sort the task set in the order of non-increasing utilization.

(2) $i := 1$; $m := 1$;

(3) $j := 1$; While $(u_i > 2/\prod_{l=1}^{k_j}(u_{j,l}+1))$ Do $j := j+1$;

(4) $k_j := k_j + 1$; $U_j := U_j + 1$; // Assign task τ_i to P_j

(5) If $(j > m)$ Then $m := j$;

(6) $i := i+1$;

(7) If $(i > n)$ Then Exit Else Goto 3;

Note that the difference between RM-FFDU and the bin-packing heuristic FFD is the usage of a different condition on a processor (or bin), the same is true between RM-FFDU and FFDUF. When the algorithm returns, the value of m is the number of

processors required by RM-FFDU to schedule the task set Σ. k_j is the number of tasks assigned on processor P_j. Clearly, RM-FFDU runs at a time complexity of $O(n \log n)$. While the RM-FFDU algorithm resembles the FFD bin-packing heuristic in assignment strategy, none of the performance results for FFD can be directly applied here, since the the utilization of a processor is constrained to be a variable in RM-FFDU, while the size of a bin remains fixed or unitary.

4 The Analysis of RM-FFDU

Although it is not difficult to design scheduling algorithms for RMMS by adopting existing bin-packing heuristics, it is rather difficult to analyze their worst-case performance; to obtain the tight bounds for these heuristics is even more so. Because of the importance of the final result, we have taken the necessary efforts to prove such result, and clarity and rigor are of our prime concern, though the number of steps involved may seem daunting. Worst-case analysis is especially essential for real-time algorithms, since missing a deadline in a hard real-time system can result in catastrophic consequences. Once its performance is known, the algorithm can be used by practitioners in applications that demand guaranteed performance, without worrying about its worst-case behavior.

We state the upper bound for RM-FFDU in the following theorem.

Theorem 2 $\Re_{RM-FFDU} \leq 5/3$

Our proof proceeds in the following fashions: Lemma 1 states a unique property possessed by the RM-FFDU schedules. Lemma 2 puts a rough upper bound on the performance of RM-FFDU. Though Lemma 2 is not essential in proving the final result, it is illustrative of the nature of the RM-FFDU schedules. The following six lemmas show that Theorem 2 holds for six different types of RM-FFDU schedules, thus leading to the final proof of the theorem after the lemmas. The examples that show the bound is tight appear at the end of this section.

In the RM-FFDU schedule, let N_i be the number of processors to each of which i tasks are assigned. Then, the total number of processors in the final RM-FFDU

schedule is given by $N = \Sigma_{i=1}^{\kappa} N_i$, where κ is the maximum number of tasks assigned to a processor, and the total number of tasks in the task set is given by $n = \Sigma_{i=1}^{\kappa} iN_i$. In the optimal schedule, let M_i be the number of processors to each of which i tasks are assigned. Then, the minimum number of processors required is given by $N_0 = \Sigma_{i=1}^{\kappa} M_i$, and similarly $n = \Sigma_{i=1}^{\kappa} iM_i$. We are trying to find the maximum of $\Re_A = \max(N/N_0)$ for any value of x. Let us define y_i to be the utilization of the first task assigned to the ith processor P_i. Then it follows from the way RM-FFDU assigns tasks to processors that $y_i \geq y_j$ if $i > j$. Where there is no confusion, we will simply use y to denote the utilization of the first task assigned to a processor.

Let x be the first item (or its utilization) assigned to the last processor in the final RM-FFDU schedule. If $x \leq \ln 2 - 3/5$, then for every busy processor except possibly the last processor, it is allocated a utilization at a level that is at least $\ln 2 - x \geq 3/5$. This is because $f(n) = n(2^{1/n} - 1) \rightarrow \ln 2$ for $n \rightarrow \infty$, and if $\sum_{i=1}^{n} C_i/T_i \leq \ln 2$, then the n tasks are always feasible with the rate-monotonic algorithm. Then $\Re_{RM-FFDU} \leq 5/3$. If $x > 1/2$, then $N = N_0$, where N_0 is the minimum number of processors required to run a task set Σ and N is the number of processors required by RM-FFDU to run the same task set Σ. For convenience, we use $\Re_A$ to denote $\Re_{RM-FFDU}$ whenever possible.

Lemma 1 *If $2^{1/(c+1)} - 1 < y \leq 2^{1/c} - 1$ for $c \geq 1$ in the completed RM-FFDU schedule, then among all processors on each of which at least c tasks are assigned, there are at most one processor to which not all the first c tasks are assigned tasks each with a utilization greater than $2^{1/(c+1)} - 1$.*

Proof. This lemma is proven by contradiction.

Suppose that there are two such processors P_i and P_j with $i < j$ such that each of them is assigned at least c tasks. Furthermore, let $\tau_{i,k}$ and $u_{i,k}$ be the task assigned to processor P_i and its utilization. Then for processor P_i, there exists at least one task $\tau_{i,m}$ with $m \leq c$ having a utilization $u_{i,m} \leq 2^{1/(c+1)} - 1$.

For processor P_i, since $2^{1/(c+1)} - 1 < y \leq 2^{1/c} - 1$ for $c \geq 1$ ($y = u_{i,1}$ by definition), all the tasks yet to be assigned after task $\tau_{i,1}$ have utilizations less than or equal to $u_{i,1}$. Furthermore, at least c tasks can be assigned on processor P_i. This is because

c	$x \leq$	$\Re_A$	c	$x \leq$	$\Re_A$
2	0.4142	2.4	5	0.1487	1.68
3	0.2599	1.92	6	0.1245	1.63
4	0.1892	1.76	∞	$\rightarrow 0$	$1/\ln 2$

Table 2: Performance of RM-FFDU for some values of x

the first c tasks are feasible with the rate-monotonic algorithm, since $u_{i,1} \leq 2^{1/c} - 1$, and the total utilization of the first c tasks is less than or equal to $c(2^{1/c} - 1)$ (recall the Liu and Layand's condition, $\sum_{i=1}^{n} u_i \leq n(2^{1/n} - 1)$, for a feasible task set).

For processor P_j with $i < j$, $\tau_{i,m}$ with $m \leq c$ must be assigned to processor P_i after the task $\tau_{j,1}$ is assigned to processor P_j. This could only happen when the first task assigned to processor P_j cannot be assigned to processor P_i, since $u_{j,1} > 2^{1/c} - 1 \geq u_{i,m}$.

Since $U_i + u_{j,1} \geq c \times (2^{1/c} - 1)$, where U_i is the total utilization assigned to processor P_i when processor P_j was first assigned the task $\tau_{j,1}$, then processor P_i must have been assigned c or more tasks each with a utilization equal to or greater than $u_{j,1} > 2^{1/(c+1)} - 1$. This is a contradiction to the assumption that there exists at least one task $\tau_{i,m}$ having a utilization $u_{i,m} \leq 2^{1/(c+1)} - 1$ with $m \leq c$ on processor P_i.

Therefore, the lemma must be true. □

Before we move on, let us obtain the upper bound (not tight) for some of the values of x. Recall that x is the first item (or its utilization) assigned to the last processor in the final RM-FFDU schedule. We will be paying close attention to all the busy processors except the last one in the final RM-FFDU schedule for their utilizations and the tasks assigned to them.

Since x is the utilization of the first task on the last processor, we can assume, without loss of generality, that the first task on the last processor is the last task in the task set after sorting. Note that the tasks following the first task on the last processor do not affect the number of processors used by RM-FFDU if they are included in the task set. Therefore, we can always assume for the RM-FFDU schedules that the utilization of any other task in the task set is equal to or larger than x. We will use

this assumption throughout the rest of the proof.

Lemma 2 *For some values of x, $\Re_A$ is given as in Table 2.*

Proof.

Since $\Re_A = 1$ for $x > 1/2$ from above, we will consider $0 < x < (2^{1/2} - 1)$ for the moment. We will divide the region $(0, 2^{1/2} - 1)$ into an infinite number of sub-regions according to $(2^{1/(c+1)} - 1) \leq x < (2^{1/c} - 1)$ and derive $\Re_A$ in term of x. For any value of x such that $(2^{1/(c+1)} - 1) \leq x < (2^{1/c} - 1)$ for $c = 2, 3, \ldots$, we need to find the minimum processor utilization among the busy processors. Let us suppose that $n_i \geq 1$ tasks are assigned to the ith processor P_i with a total utilization of U_i,

Case 1: If $n_i \geq c$, i.e., each of the n_i tasks assigned on the ith processor has a utilization no less than x, then $U_i \geq n_i x \geq c(2^{1/(c+1)} - 1)$.

Case 2: If $n_i < c$, then $c \geq 2$ and $x > n_i(2^{1/n_i} - 1) - U_i > c(2^{1/c} - 1) - U_i$. Since $(2^{1/c} - 1) > x$, we have $U_i > (c-1)(2^{1/c} - 1)$.

For both cases, every busy processor except the last one has a utilization no smaller than $(c-1)(2^{1/c} - 1)$ for $c \geq 2$. Suppose that N processors are used in the final RM-FFDU schedule, then the total utilization of the N processors is at least $N(c-1)(2^{1/c} - 1)$. Since the utilization of a processor cannot exceed one in the optimal schedule, the optimal number N_0 of processors required for the same task set must be at least $N(c-1)(2^{1/c} - 1)$, i.e, $N_0 \geq N(c-1)(2^{1/c} - 1)$. Hence $N/N_0 \leq [(c-1) \times (2^{1/n} - 1)]^{-1}$, i.e., $\Re_A \leq [(c-1) \times (2^{1/n} - 1)]^{-1}$ for $c \geq 2$. The values of $\Re_A$ are given in Table 2 for some values of c. □

For those processors to each of which n tasks are assigned, their minimum utilization can be determined by the following method: let $u_1, u_2, \ldots, u_n$ be the utilizations of the n tasks. Since $x > 2/\prod_{i=1}^{n}(1 + u_i) - 1$ according to the UO condition, the minimum of $U = \Sigma_{i=1}^{n} u_i$ is achieved at $U = n \times \{[2/(1+x)]^{1/n} - 1\}$ when $u_1 = u_2 = \ldots = u_n = [2/(1+x)]^{1/n} - 1$.

In the subsequent lemmas, we will prove that $\Re_A \leq 5/3$ with regard to x. We divide the range of values x can assume into several intervals and prove that $\Re_A \leq 5/3$ for each interval:

$x \in (1/3, 1/2]$, $x \in (1/4, 1/3]$, $x \in (1/5, 1/4]$, $x \in (2^{1/4} - 1, 1/5]$, $x \in (1/6, 2^{1/4} - 1]$, $x \in (5(2^{1/5} - 1) - 3/5, 1/6]$, and $x \in (6(2^{1/6} - 1) - 3/5, 5(2^{1/5} - 1) - 3/5]$.

The final proof of Theorem 2 appears after the lemmas. Recall the meanings of N_i and M_i and that κ is the maximum number of tasks assigned to a processor among all processors in the RM-FFDU schedule.

Lemma 3 *If* $x \in (1/3, 1/2]$, *then* $\Re_A \leq 3/2$.

Proof. Since $x > 1/3$, a processor cannot be assigned more than 2 tasks, i.e., $\kappa \leq 2$. Each processor is assigned one or two tasks in either the RM-FFDU schedule or the optimal schedule.

Let n be the total number of tasks in a task set. The optimal number of processors required is given by $N_0 = \Sigma_{i=1}^{2} M_i$, and the number of tasks is determined by $n = \Sigma_{i=1}^{2} i M_i$. In the RM-FFDU schedule, $N = \Sigma_{i=1}^{2} N_i$ and $n = \Sigma_{i=1}^{2} i N_i$.

Then the ratio N/N_0 is maximized when $M_1 = 0$ and $M_2 = n/2$. Hence, the maximum value $\Re_A$ is achieved at $3/2$, i.e., $\Re_A = 3/2$. □

Lemma 4 *If* $x \in (1/4, 1/3]$, *then* $\Re_A \leq 3/2$.

Proof. Since $x > 1/4$, a processor can be assigned no more than 3 tasks, i.e., $\kappa \leq 3$. In the RM-FFDU schedule, let us consider all the processors to each of which one task is assigned. We want to find out the minimum utilization of such task. Let u be the utilization of such task. Since $x \leq 1/3$, according to the UO condition, $u > (1 - 1/3)/(1 + 1/3) = 1/2$. In other words, among all the processors to each of which one task is assigned, the utilization of each of these tasks must be greater than $1/2$. Therefore, if there are N_1 such processors in the RM-FFDU schedule, then at least N_1 processors are needed in the optimal schedule, because any of the two tasks cannot be assigned to a processor.

Recall that $N = N_1 + N_2 + N_3$ and $N_0 = N_1 + N_2 + N_3$.

Case 1: $N_1 = 0$. Since each processor in the RM-FFDU schedule is assigned at least two tasks and each processor in the optimal schedule cannot be assigned more than four tasks (since $u_i > 1/4$), then $\Re_A$ is maximized when $N_3 = 0$ and $M_2 = 0$. Thus, $\Re_A \leq 3/2$.

Case 2: $N_2 = 0$. This implies that each processor in the RM-FFDU schedule is assigned either one or three tasks. The utilization of each of the N_1 tasks, each of which is assigned to one of the N_1 processors, is greater than $1/2$ as mentioned above. For these N_1 tasks, any pair of them cannot be assigned to a processor in the optimal schedule. Furthermore, for those N_1 processors in the optimal schedule, each can be assigned at most two tasks. This is because each task has a utilization greater than $1/4$ and one task has a utilization greater than $1/2$. Therefore the maximum of $\Re_A$ is achieved when $N_1 = 3N_3$ such that $\Re_A \leq 4/3$.

Case 3: $N_3 = 0$. For similar reasons, we can prove that $\Re_A \leq 3/2$.

Case 4: $N_i \neq 0$ for $i = 1, 2, 3$. We claim that $\Re_A \leq 3/2$. Since there are N_1 tasks, each with a utilization greater than $1/2$, these tasks must use N_1 processors in the optimal schedule. Suppose that each of these N_1 processor is assigned two tasks. Then the minimum number of processors required in the optimal schedule is at least $N_1+(2N_2+3N_3-N_1)/3$. Therefore, $\Re_A \leq (N_1+N_2+N_3)/[N_1+(2N_2+3N_3-N_1)/3] \leq 3/2$. □

Lemma 5 *If* $x \in (1/5, 1/4]$, *then* $\Re_A \leq 3/2$.

Proof. Since $x > 1/5$, a processor is assigned at most four tasks, i.e., $\kappa \leq 4$.

For similar reasons used in the proof of Lemma 4, we need to find out the minimum utilization of a task assigned to a particular processor and the minimum utilization of the processor. We divide the N processors in the RM-FFDU schedule into κ groups, each of which has N_i processors, where $1 \leq i \leq \kappa$.

For those processors to each of which one task is assigned, the utilization of the task is greater than $(1 - 1/4)/(1 + 1/4) = 3/5$. Then for any of the N_1 processor, its minimum utilization U is given by $U > 3/5$.

For those processors to each of which two tasks (with utilizations u_1 and u_2) are assigned, the minimum of $(u_1 + u_2)$ is achieved at $2[\sqrt{2/(1+x)} - 1]$ when $u_1 = u_2 = \sqrt{2/(1+x)} - 1$. Then, for $x = 1/4$, $u_1 = u_2 = \sqrt{8/5} - 1 = 0.2649$, and $U = 0.529$. Note that for $x > 1/5$, $U > 0.529$. Two more tasks could be assigned on these processors in the optimal schedule, since $2x + U \leq 1$ for some $x > 1/5$.

$W(u) =$	$u \in$
0	$(0, 1/5]$
1/3	$(1/5, \sqrt{8/5} - 1]$
1/2	$(\sqrt{8/5} - 1, \sqrt{2} - 1]$
2/3	$(\sqrt{2} - 1, 3/5]$
1	$(3/5, 1]$

Table 3: Weighting Function for $x \in (1/5, 1/4]$

For those processors to each of which three or four tasks are assigned, their minimum total utilization is determined by $U \geq 3x > 0.6$, when $u_i > 1/5$.

In the following, we define a function that maps the utilization of a task to a value that is in the range of 0 and 1, as given in Table 3. We call that value the weight of the task, and the sum of the weights of the tasks assigned to a processor the weight of the processor. The weighting function is designed in such a way that for every processor in the RM-FFDU schedule, its weight is equal to or greater than one. At the meantime, the weight of a processor in the optimal schedule is no greater than 5/3. We first claim that for any processor P in the completed RM-FFDU schedule, the total weight of processor P is equal to or greater than one, i.e., $W(P) = \Sigma_{i=1}^{\kappa} \geq 1$, where κ is the number of tasks assigned to processor P.

Recall that in the completed RM-FFDU schedule, any task utilization y must be equal to or greater than x, i.e., $y \geq x$. Let us consider a processor to which a task with a utilization of y is first assigned.

Case 1: $1/5 < y \leq \sqrt{8/5} - 1$. Then the processor must be assigned at least three tasks each with a utilization greater than 1/5. Therefore, $W(P) \geq 3 \times 1/3 = 1$.

Case 2: $\sqrt{8/5} - 1 < y \leq \sqrt{2} - 1$. Then the processor must be assigned at least two tasks. Furthermore, except for possibly one processor by Lemma 1, each of the first two tasks must have a utilization greater than $\sqrt{8/5} - 1$. Therefore, $W(P) \geq 1$.

Case 3: $\sqrt{2} - 1 < y \leq 3/5$. Then the processor must be assigned at least two

tasks. Since the second task must be a task with a utilization greater than 1/5, we have $W(P) \geq 1$.

Case 4: $3/5 < y \leq 1$. Then $W(P) \geq 1$ by definition.

Next we prove that for any processor P in the optimal schedule, $W(P) \leq 3/2$.

Let us assume that a processor in the optimal schedule is assigned m tasks with their utilizations as $u_1 \geq u_2 \geq \ldots \geq u_m$.

Case I: $u_1 < \sqrt{8/5} - 1$. Then at most four tasks each with a utilization greater than 1/5 can be assigned on it. Therefore, $W(P) \geq 4/3$.

Case II: $\sqrt{8/5} - 1 < u_1 < \sqrt{2} - 1$ and $\sqrt{8/5} - 1 \leq u_2$. Then at most one more task can be assigned to the processor. If $u_3 \leq \sqrt{8/5} - 1$, then $W(P) \leq 1/2 + 1/2 + 1/3 = 4/3$. If $u_3 > \sqrt{8/5} - 1$, then $W(P) \leq 1/2 + 1/2 + 1/2 = 3/2$.

Case III: $\sqrt{8/5} - 1 < u_1 < \sqrt{2} - 1$ and $\sqrt{8/5} - 1 > u_2$. Then at most two more tasks are assigned to the processor. Therefore, $W(P) \leq 1/2 + 1/3 + 1/3 + 1/3 = 3/2$.

Case IV: $\sqrt{2} - 1 < u_1 < 3/5$ and $u_2 > \sqrt{2} - 1$. Then no more task with a utilization greater than 1/5 can be assigned to the processor. Therefore, $W(P) \leq 2/3 + 2/3 = 4/3$.

Case V: $\sqrt{2} - 1 < u_1 < 3/5$ and $\sqrt{8/5} - 1 < u_2 \leq \sqrt{2} - 1$. Then no more task with a utilization greater than 1/5 can be assigned to the processor. Therefore, $W(P) \leq 2/3 + 1/2 = 7/6$.

Case VI: $\sqrt{2} - 1 < u_1 < 3/5$ and $u_2 \leq \sqrt{8/5} - 1$. Then at most one more task with a utilization greater than 1/5 can be assigned to the processor. Therefore, $W(P) \leq 2/3 + 1/3 + 1/3 = 4/3$.

Case VII: $3/5 < u_1 < 1$. Then at most one more task with a utilization greater than 1/5 can be assigned to the same processor. Furthermore, $u_2 < \sqrt{2} - 1$. Then $W(P) \leq 1 + 1/2 = 3/2$.

Let N and N_0 be number of processors required by RM-FFDU and the minimum number of processors required to schedule a given set Σ of n tasks, respectively. Then the total weight of the task set is given by $\Sigma_{i=1}^{n} W(u_i)$. Since, except for one processor, $W(P) \geq 1$ for every processor in the RM-FFDU schedule, then $\Sigma_{i=1}^{n} W(u_i) \geq N - 1$. Since $W(P) \leq 3/2$ for every processor in the optimal schedule, $N_0 \times 3/2 \geq \Sigma_{i=1}^{n} W(u_i)$. Therefore, $\Re_A \leq 3/2$. □

$W(u) =$	$u \in$
0	$(0, 2^{1/4} - 1]$
1/3	$(2^{1/4} - 1, \sqrt{5/3} - 1]$
1/2	$(\sqrt{5/3} - 1, \sqrt{2} - 1]$
2/3	$(\sqrt{2} - 1, 2/3]$
1	$(2/3, 1]$

Table 4: Weighting Function for $x \in (2^{1/4} - 1, 1/5]$

Lemma 6 *If* $x \in (2^{1/4} - 1, 1/5]$, *then* $\Re_A \leq 5/3$.

Proof. Since $x > 2^{1/4} - 1 = 0.1892$, a processor is assigned at most five tasks, i.e., $\kappa \leq 5$.

For those processors to each of which one task is assigned, the utilization of each task is greater than $(1 - 1/5)/(1 + 1/5) = 2/3$.

For the processor to which two tasks are assigned, the minimum of $u_1 + u_2$ is achieved at $U = 2 \times [\sqrt{2/(1+x)} - 1]$ when $u_1 = u_2 = \sqrt{2/(1+x)} - 1$. Then, for $x = 1/5$, $u_1 = u_2 = \sqrt{5/3} - 1 \approx 0.29$, and $U = 2[\sqrt{5/3} - 1] = 0.58$. Note that for $x < 1/5$, $U > 0.58$.

For a processor to which three tasks are assigned, their minimum utilization is achieved at $U = 3 \times \{[2/(1+x)]^{1/3} - 1\}$ when $u_i = [2/(1+x)]^{1/3} - 1$. We want to fix x such that $x \leq [2/(1+x)]^{1/3} - 1$. Solving the inequality $x \leq [2/(1+x)]^{1/3} - 1$ yields $x \leq 2^{1/4} - 1$. In other words, for every processor to which three tasks are assigned in the completed RM-FFDU schedule, their total utilization is greater than $3 \times (2^{1/4} - 1) \approx 0.5676$.

In the following, we define a function that maps the utilization of a task to a value that is in the range of 0 and 1, as given by Table 4. The weighting function is designed in such a way that for every processor in the RM-FFDU schedule, its weight is equal to or greater than 1. At the meantime, the weight of a processor in the optimal schedule is no greater than 5/3.

We first prove that for every processor P in the completed RM-FFDU schedule, $W(P) \geq 1$.

Recall that in the completed RM-FFDU schedule, any task utilization y must be equal to or greater than x, i.e., $y \geq x$. Let us consider a processor to which a task with a utilization of y is first assigned.

Case 1: $2^{1/4} - 1 < y \leq \sqrt{5/3} - 1$. Then the processor must be assigned at least three tasks. Therefore, $W(P) \geq 1/3 + 1/3 + 1/3 = 1$.

Case 2: $\sqrt{5/3} - 1 < y \leq \sqrt{2} - 1$. Then the processor must be assigned at least two tasks. Furthermore, except for one processor by Lemma 1, each of the first two tasks must have a utilization greater than $\sqrt{5/3} - 1$. Therefore, $W(P) \geq 1/2 + 1/2 = 1$.

Case 3: $\sqrt{2} - 1 < y \leq 2/3$. Then the processor must be assigned at least two tasks. Since the second task must be a task with a utilization greater than $\sqrt{2} - 1$, we have $W(P) \geq 2/3 + 1/3 = 1$.

Case 4: $2/3 < y \leq 1$, $W(P) \geq 1$ by definition.

We then prove that for any processor in the optimal schedule, $W(P) \leq 5/3$.

Let us assume that a processor in the optimal schedule is assigned m tasks with their utilizations as $u_1 \geq u_2 \geq \ldots \geq u_m$.

Case I: $u_1 < \sqrt{5/3} - 1$. Then at most five tasks each with a utilization greater than $(2^{1/4} - 1)$ can be assigned on a processor. Therefore, $W(P) \leq 5/3$.

Case II: $\sqrt{5/3} - 1 < u_1 < \sqrt{2} - 1$ and $\sqrt{5/3} - 1 \leq u_2$. Then at most two more tasks can be assigned to the processor. If $u_3 \leq \sqrt{5/3} - 1$, then $W(P) \leq 1/2 + 1/2 + 1/3 + 1/3 = 5/3$. If $u_3 > \sqrt{5/3} - 1$, then $W(P) \leq 1/2 + 1/2 + 1/2 = 3/2$.

Case III: $\sqrt{5/3} - 1 < u_1 < \sqrt{2} - 1$ and $\sqrt{5/3} - 1 \geq u_2$. Then at most two more tasks each with a utilization less than $(\sqrt{5/3} - 1)$ can be assigned to the processor. Therefore, $W(P) \leq 1/2 + 1/3 + 1/3 + 1/3 = 3/2$.

Case IV: $\sqrt{2} - 1 < u_1 < 2/3$ and $u_2 > \sqrt{2} - 1$. Then no more task with a utilization greater than $2^{1/4} - 1$ can be assigned to the processor. Therefore, $W(P) \leq 2/3 + 2/3 = 4/3$.

If $\sqrt{2} - 1 < u_1 < 2/3$ and $\sqrt{5/3} - 1 < u_2 < \sqrt{2} - 1$, then at most one task with a utilization greater than $2^{1/4} - 1$ and less than $\sqrt{2} - 1$ can be assigned to the processor.

Therefore, $W(P) \leq 2/3 + 1/2 + 1/2 = 5/3$.

If $\sqrt{2} - 1 < u_1 < 2/3$ and $u_2 < \sqrt{5/3} - 1$, then at most two more task with a utilization greater than $2^{1/4} - 1$ can be assigned to the processor. Therefore, $W(P) \leq 2/3 + 1/3 + 1/3 + 1/3 = 5/3$.

Case V: $2/3 < u_1 < 1$. Then at most one more task with a utilization greater than $2^{1/4} - 1$ can be assigned to the same processor. Furthermore, $u_2 < \sqrt{2} - 1$. Then $W(P) \leq 1 + 1/2 = 3/2$.

Let N and N_0 be number of processors required by RM-FFDU and the minimum number of processors required to schedule a given set Σ of n tasks, respectively. Then the total weight of the task set is given by $\Sigma_{i=1}^{n} W(u_i)$. Since, except for one processor, $W(P) \geq 1$ for every processor in the RM-FFDU schedule, then $\Sigma_{i=1}^{n} W(u_i) \geq N - 1$. Since $W(P) \leq 5/3$ for every processor in the optimal schedule, $N_0 \times 5/3 \geq \Sigma_{i=1}^{n} W(u_i)$. Therefore, $\Re_A \leq 5/3$. □

Lemma 7 *If* $x \in (1/6, 2^{1/4} - 1]$, *then* $\Re_A \leq 5/3$.

Proof. Since $x > 1/6$, a processor is assigned at most five tasks, i.e., $\kappa \leq 5$.

For those processors to each of which one task is assigned, the utilization of each task is greater than $[1 - (2^{1/4} - 1)]/(1 + 2^{1/4} - 1) = 2^{3/4} - 1 \approx 0.68$.

For the processor to which two tasks are assigned, the minimum of $u_1 + u_2$ is achieved at $U = 2 \times [\sqrt{2/(1+x)} - 1]$ when $u_1 = u_2 = \sqrt{2/(1+x)} - 1$. Then, for $x = 2^{1/4} - 1 \approx 0.1892$, $u_1 = u_2 = 2^{3/8} - 1 \approx 0.297$ and $U = 2 \times (2^{3/8} - 1) \approx 0.594$. Note that for $x < 2^{1/4} - 1$, $U > 0.594$.

For a processor to which three tasks are assigned, their minimum utilization is achieved at $U = 3 \times \{[2/(1+x)]^{1/3} - 1\}$ when $u_i = [2/(1+x)]^{1/3} - 1$. We want to fix x such that $x \leq [2/(1+x)]^{1/3} - 1$. Solving the inequality $x \leq [2/(1+x)]^{1/3} - 1$ yields $x \leq 2^{1/4} - 1$. In other words, for every processor to which three tasks are assigned in the completed RM-FFDU schedule, their total utilization is greater than $3 \times (2^{1/4} - 1) \approx 0.5676$.

For $y < 2^{1/4} - 1$, each processor must be assigned at least four tasks each with a utilization less than $(2^{1/4} - 1)$.

$W(u) =$	$u \in$
0	$(0, 1/6]$
1/3	$(1/6, 2^{3/8} - 1]$
1/2	$(2^{3/8} - 1, \sqrt{2} - 1]$
2/3	$(\sqrt{2} - 1, 2^{3/4} - 1]$
1	$(2^{3/4} - 1, 1]$

Table 5: Weighting Function for $x \in (1/6, 2^{1/4} - 1]$

In the following, we define a function that maps the utilization of a task to a value that is in the range of 0 and 1, as given by Table 5. The weighting function is designed in such a way that for every processor in the RM-FFDU schedule, its weight is equal to or greater than 1, and at the meantime, the weight of a processor in the optimal schedule is no greater than 5/3.

We first prove that for every processor P in the completed RM-FFDU schedule, $W(P) \geq 1$.

Recall that in the completed RM-FFDU schedule, any task utilization y must be equal to or greater than x, i.e., $y \geq x$. Let us consider a processor to which a task with a utilization of y is first assigned.

Case 1: $1/6 < y \leq 2^{1/4} - 1$. Then the processor must be assigned at least four tasks. Therefore, $W(P) \geq 4 \times 1/3 > 1$.

Case 2: $2^{1/4} - 1 < y \leq 2^{3/8} - 1$. Then the processor must be assigned at least three tasks. Therefore, $W(P) \geq 1/3 + 1/3 + 1/3 = 1$.

Case 3: $2^{3/8} - 1 < y \leq \sqrt{2} - 1$. Then the processor must be assigned at least two tasks. Furthermore, except for one processor by Lemma 1, each of the first two tasks must have a utilization greater than $2^{3/8} - 1$. Therefore, $W(P) \geq 1/2 + 1/2 = 1$.

Case 4: $\sqrt{2} - 1 < y \leq 2^{3/4} - 1$. Then the processor must be assigned at least two tasks. Since the second task must be a task with a utilization greater than $\sqrt{2} - 1$, we have $W(P) \geq 2/3 + 1/3 = 1$.

Case 5: $2^{3/4} - 1 < y \leq 1$. Then $W(P) \geq 1$ by definition.

We then prove that for any processor in the optimal schedule, $W(P) \leq 5/3$.

Let us assume that a processor in the optimal schedule is assigned m tasks with their utilizations as $u_1 \geq u_2 \geq \ldots \geq u_m$.

Case I: $1/6 < u_1 < 2^{3/8} - 1$. Then at most four tasks each with a utilization greater than $1/6$ and less than $2^{3/8} - 1$ can be assigned on a processor. Therefore, $W(P) \leq 5 \times 1/3 = 5/3$.

Case II: $2^{3/8} - 1 < u_1 < \sqrt{2} - 1$ and $2^{3/8} - 1 \leq u_2$. Then at most two more tasks can be assigned to the processor. If $u_3 \leq 2^{3/8} - 1$, then $W(P) \leq 1/2 + 1/2 + 1/3 + 1/3 = 5/3$. If $u_3 > 2^{3/8} - 1$, then no more task with a utilization greater than $1/6$ can be assigned to the processor, and thus $W(P) \leq 1/2 + 1/2 + 1/2 = 3/2$.

Case III: $2^{3/8} - 1 < u_1 < \sqrt{2} - 1$ and $u_2 < 2^{3/8} - 1$. Then at most two more tasks each with a utilization less than $2^{3/8} - 1$ can be assigned to the processor. Therefore, $W(P) \leq 1/2 + 1/3 + 1/3 + 1/3 = 3/2$.

Case IV: $\sqrt{2} - 1 < u_1 < 2^{3/4} - 1$ and $\sqrt{2} - 1 < u_2$. Then at most one task with a utilization greater than $1/6$ and less than $2^{1/4} - 1$ can be assigned to the processor. Therefore, $W(P) \leq 2/3 + 2/3 + 1/4 < 5/3$.

If $\sqrt{2} - 1 < u_1 < 2^{3/4} - 1$ and $2^{3/8} - 1 < u_2 < \sqrt{2} - 1$, then at most one task with a utilization greater than $1/6$ and less than $2^{3/4} - 1$ can be assigned to the processor. Therefore, $W(P) \leq 2/3 + 1/2 + 1/3 = 3/2$.

If $\sqrt{2} - 1 < u_1 < 2^{3/4} - 1$ and $u_2 < 2^{3/8} - 1$, then at most two more task with a utilization greater than $2^{1/4} - 1$ can be assigned to the processor. Therefore, $W(P) \leq 2/3 + 1/3 + 1/3 + 1/3 = 5/3$.

Case V: $2^{3/4} - 1 < u_1 < 1$. Then at most one more task with a utilization greater than $1/6$ and less than $2^{3/8} - 1$ can be assigned to the same processor. Then $W(P) \leq 1 + 1/3 = 4/3$.

Let N and N_0 be number of processors required by RM-FFDU and the minimum number of processors required to schedule a given set Σ of n tasks, respectively. Then the total weight of the task set is given by $\Sigma_{i=1}^{n} W(u_i)$. Since, possibly except for one processor, $W(P) \geq 1$ for every processor in the RM-FFDU schedule, then $\Sigma_{i=1}^{n} W(u_i) \geq$

$N - 1$. Since $W(P) \leq 5/3$ for every processor in the optimal schedule, $N_0 \times 5/3 \geq \Sigma_{i=1}^{n} W(u_i)$. Therefore, $\Re_A \leq 5/3$. □

Lemma 8 *If* $x \in (5(2^{1/5} - 1) - 3/5, 1/6]$, *then* $\Re_{RM-FFDU} \leq 5/3$.

Proof. Since $5(2^{1/5} - 1) - 3/5 \approx 0.14349 > 1/7$, a processor can be assigned at most six tasks, i.e., $\kappa \leq 6$. For convenience, let us denote $\delta = 5(2^{1/5} - 1) - 3/5$.

For those processors to each of which one task is assigned, the utilization of each task is greater than $(1 - 1/6)/(1 + 1/6) = 5/7 \approx 0.71$.

For the processor to which two tasks are assigned, the minimum of $u_1 + u_2$ is achieved at $U = 2 \times [\sqrt{2/(1+x)} - 1]$ when $u_1 = u_2 = \sqrt{2/(1+x)} - 1$. Then, for $x = 1/6$, $u_1 = u_2 = \sqrt{12/7} - 1 \approx 0.31$ and $U = 2[\sqrt{12/7} - 1] = 0.62$. Note that for $x < 1/6$, $U > 0.62$.

For a processor to which three tasks are assigned, their minimum utilization is achieved at $U = 3 \times \{[2/(1+x)]^{1/3} - 1\}$ when $u_i = [2/(1+x)]^{1/3} - 1$. Then, for $x = 1/6$, $u_1 = u_2 = u_3 = (12/7)^{1/3} - 1 \approx 0.197$ and $U = 3[(12/7)^{1/3} - 1] \approx 0.59$. Note that for $x < 1/6$, $U > 0.59$.

For a processor to which four tasks are assigned, their minimum utilization is achieved at $U = 4 \times \{[2/(1+x)]^{1/4} - 1\}$ when $u_i = [2/(1+x)]^{1/4} - 1$. We want to fix x such that $x \leq [2/(1+x)]^{1/4} - 1$. Solving the inequality $x \leq [2/(1+x)]^{1/4} - 1$ yields $x \leq 2^{1/5} - 1$. In other words, for every processor to which four tasks are assigned in the completed RM-FFDU schedule, their total utilization is greater than $4 \times (2^{1/5} - 1) \approx 0.595$.

As having done so above, we define a function that maps the utilization of a task to a value that is in the range of 0 and 1, as given by Table 6.

We first prove that for every processor P in the completed RM-FFDU schedule, $W(P) \geq 1$.

Recall that in the completed RM-FFDU schedule, any task utilization y must be equal to or greater than x, i.e., $y \geq x$. Let us consider a processor to which a task with a utilization of y is first assigned.

Case 1: $y < (12/7)^{1/3} - 1$. Except for the last processor, the processor must be

$W(u) =$	$u \in$
0	$(0, \delta]$
1/4	$(\delta, (12/7)^{1/3} - 1]$
1/3	$((12/7)^{1/3} - 1, 2^{1/3} - 1]$
3/8	$(2^{1/3} - 1, \sqrt{12/7} - 1]$
1/2	$(\sqrt{12/7} - 1, \sqrt{2} - 1]$
2/3	$(\sqrt{2} - 1, (12/7)^{2/3} - 1]$
3/4	$((12/7)^{2/3} - 1, 5/7]$
1	$(5/7, 1]$

Table 6: Weighting Function for $x \in (\delta, 1/6]$

assigned at least four tasks each with a utilization less than $(12/7)^{1/3} - 1$ but greater than δ. Therefore, $W(P) \geq 4 \times 1/4 = 1$.

Case 2: $(12/7)^{1/3} - 1 < y \leq 2^{1/3} - 1$. Except for one processor by Lemma 1, the processor must be assigned at least three tasks each with a utilization u such that $(12/7)^{1/3} - 1 < u \leq 2^{1/3} - 1$. Therefore, $W(P) \geq 3 \times 1/3 = 1$.

Case 3: $2^{1/3} - 1 < y \leq \sqrt{12/7} - 1$. Except for one processor by Lemma 1, the processor must be assigned at least three tasks. Furthermore, each of the first two tasks must have a utilization greater than $2^{1/3} - 1$. Therefore, $W(P) \geq 3/8 + 3/8 + 1/4 = 1$.

Case 4: $\sqrt{12/7} - 1 < y \leq \sqrt{2} - 1$. Except for one processor by Lemma 1, the processor must be assigned at least two tasks, each of which must have a utilization greater than $\sqrt{12/7} - 1$. Therefore we have $W(P) \geq 1/2 + 1/2 = 1$.

Case 5: $\sqrt{2} - 1 < y \leq (12/7)^{2/3} - 1 \approx 0.432$. Since $x \in (\delta, 1/6]$, the processor must be assigned at least two tasks each with a utilization $\delta < u \leq (12/7)^{2/3} - 1$. If the utilization of the second task is greater than $(12/7)^{1/3} - 1$, then $W(P) \geq 2/3 + 1/3 = 1$. If the utilization u_2 of the second task is equal to or less than $(12/7)^{1/3} - 1$, then one more task with a utilization $u_3 \in (\delta, 1/6]$ must be assigned on the processor. This is because

$2/(\{1+[(12/7)^{1/3}-1]\}1+[(12/7)^{2/3}-1])-1 = 7/6-1 = 1/6 \geq u_3$.

Then $W(P) \geq 2/3+1/4+1/4 > 1$.

Case 6: $(12/7)^{2/3}-1 < y \leq 5/7$. Except for one processor by Lemma 1, the processor must be assigned at least two tasks. The second task must have a utilization greater than δ. Therefore we have $W(P) \geq 3/4+1/4 = 1$.

Case 7: $5/7 < y \leq 1$. $W(P) \geq 1$ by definition.

We then prove that for any processor in the optimal schedule, $W(P) \leq 5/3$.

Let us assume that a processor P in the optimal schedule is assigned m tasks with their utilizations as $u_1 \geq u_2 \geq \ldots \geq u_m \geq x$.

Case I: $\delta < u_1 \leq (12/7)^{1/3}-1$. Then at most six tasks each with a utilization greater than δ (and $\leq u_1$) can be assigned on a processor. Therefore, $W(P) \leq 6/4 < 5/3$.

Case II: $(12/7)^{1/3}-1 < u_1 \leq 2^{1/3}-1$. There are three sub-cases to consider. If $u_2 \leq (12/7)^{1/3}-1$, then at most four more tasks each with a utilization greater than δ can be assigned to the processor, i.e., $m \leq 6$. Then $W(P) \leq 1/3+5\times 1/4 < 5/3$.

If $u_2 > (12/7)^{1/3}-1$ and $u_3 \leq (12/7)^{1/3}-1$, then at most three more tasks each with a utilization greater than δ can be assigned to the processor, i.e., $m \leq 6$. Then $W(P) \leq 1/3+1/3+4\times 1/4 = 5/3$.

If $u_3 > (12/7)^{1/3}-1$ and $u_4 \leq (12/7)^{1/3}-1$, then at most one more task with a utilization greater than δ can be assigned to the processor, i.e., $m \leq 5$. This is because $3\times[(12/7)^{1/3}-1]+3\times\delta > 1$. Then $W(P) \leq 3\times 1/3+2\times 1/4 < 5/3$.

If $u_4 > (12/7)^{1/3}-1$ and $u_5 \leq (12/7)^{1/3}-1$, then no more task with a utilization greater than δ can be assigned to the processor, i.e., $m \leq 5$. Then $W(P) \leq 4\times 1/3+1/4 < 5/3$. If $u_5 > (12/7)^{1/3}-1$, then $W(P) \leq 5\times 1/3 = 5/3$.

Case III: $2^{1/3}-1 < u_1 \leq \sqrt{12/7}-1$. There are several sub-cases to consider. If $u_2 \leq (12/7)^{1/3}-1$, then at most four more tasks each with a utilization greater than δ (and less than $(12/7)^{1/3}-1$) can be assigned to the processor, i.e., $m \leq 6$. Then $W(P) \leq 3/8+5\times 1/4 < 5/3$.

If $(12/7)^{1/3}-1 < u_2 \leq 2^{1/3}-1$ and $u_3 \leq (12/7)^{1/3}-1$, then at most two more tasks each with a utilization greater than δ can be assigned to the processor, i.e., $m \leq 5$. Then $W(P) \leq 3/8+1/3+3\times 1/4 < 5/3$.

If $u_2 \leq 2^{1/3} - 1$, $(12/7)^{1/3} - 1 < u_3 \leq 2^{1/3} - 1$, and $u_4 \leq (12/7)^{1/3} - 1$, then at most one more task with a utilization greater than δ can be assigned to the processor, i.e., $m \leq 5$. Then $W(P) \leq 3/8 + 2 \times 1/3 + 2 \times 1/4 < 5/3$.

If $u_3 \leq 2^{1/3} - 1$, $(12/7)^{1/3} - 1 < u_4 \leq 2^{1/3} - 1$, and $u_5 \leq (12/7)^{1/3} - 1$, then no more task with a utilization greater than δ can be assigned to the processor, i.e., $m \leq 5$. Then $W(P) \leq 3/8 + 3 \times 1/3 + 1/4 < 5/3$.

If $2^{1/3} - 1 < u_2$ and $(12/7)^{1/3} - 1 < u_3 \leq 2^{1/3} - 1$, then at most one more task with a utilization greater than δ can be assigned to the processor, i.e., $m \leq 4$. Therefore $W(P) \leq 2 \times 3/8 + 1/3 + 1/4 < 5/3$. If $2^{1/3} - 1 < u_3$, then at most more task with a utilization greater than δ and less than $2^{1/3} - 1$ can be assigned to the processor, i.e., $m \leq 4$. Therefore $W(P) \leq 3 \times 3/8 + 1/3 < 5/3$.

Case IV: $\sqrt{12/7} - 1 < u_1 \leq \sqrt{2} - 1$. There are several sub-cases to consider. If $u_2 \leq (12/7)^{1/3} - 1$, then at most three more tasks each with a utilization greater than δ (and less than $(12/7)^{1/3} - 1$) can be assigned to the processor, i.e., $m \leq 5$, since $\sqrt{12/7} - 1 + 5 \times \delta > 1$. Then $W(P) \leq 1/2 + 4 \times 1/4 < 5/3$.

If $(12/7)^{1/3} - 1 < u_2 \leq 2^{1/3} - 1$ and $u_3 \leq (12/7)^{1/3} - 1$, then at most two more tasks each with a utilization greater than δ can be assigned to the processor, i.e., $m \leq 5$, since $\sqrt{12/7} - 1 + (12/7)^{1/3} - 1 + 4 \times \delta > 1$. Then $W(P) \leq 1/2 + 1/3 + 3 \times 1/4 < 5/3$.

If $u_2 \leq 2^{1/3} - 1$ and $(12/7)^{1/3} - 1 < u_3 \leq 2^{1/3} - 1$, then at most two more tasks each with a utilization greater than δ and less than $(12/7)^{1/3} - 1$ can be assigned to the processor, i.e., $m \leq 5$. Then $W(P) \leq 1/2 + 2 \times 1/3 + 2 \times 1/4 = 5/3$.

If $u_2 \leq 2^{1/3} - 1$ and $(12/7)^{1/3} - 1 < u_4 \leq 2^{1/3} - 1$, then no more task with a utilization greater than δ can be assigned to the processor, i.e., $m \leq 4$. Then $W(P) \leq 1/2 + 3 \times 1/3 < 5/3$.

If $2^{1/3} - 1 < u_2 \leq \sqrt{12/7} - 1$ and $u_3 \leq (12/7)^{1/3} - 1$, then at most two more tasks each with a utilization greater than δ can be assigned to the processor, i.e., $m \leq 5$. Therefore $W(P) \leq 1/2 + 3/8 + 3 \times 1/4 < 5/3$. If $2^{1/3} - 1 < u_2 \leq \sqrt{12/7} - 1$ and $(12/7)^{1/3} - 1 < u_3 \leq 2^{1/3} - 1$, then at most one more task with a utilization greater than δ and less than $(12/7)^{1/3} - 1$ can be assigned to the processor, i.e., $m \leq 4$. Therefore $W(P) \leq 1/2 + 3/8 + 1/3 + 1/4 < 5/3$. If $2^{1/3} - 1 < u_2 \leq \sqrt{12/7} - 1$ and $(12/7)^{1/3} - 1 <$

$u_4 \leq u_3 \leq 2^{1/3} - 1$, then no more task with a utilization greater than δ can be assigned to the processor, i.e., $m \leq 4$. Therefore $W(P) \leq 1/2 + 3/8 + 1/3 + 1/3 < 5/3$. If $u_2 \leq \sqrt{12/7} - 1$ and $2^{1/3} - 1 < u_3 \leq \sqrt{12/7} - 1$, then at most one more task with a utilization greater than δ and less than$(12/7)^{1/3} - 1$ can be assigned to the processor, i.e., $m \leq 4$. Therefore $W(P) \leq 1/2 + 2 \times 3/8 + 1/4 < 5/3$.

If $\sqrt{12/7} - 1 < u_2 \leq \sqrt{2} - 1$ and $u_3 \leq (12/7)^{1/3} - 1$, then at most one more task with a utilization greater than δ can be assigned to the processor, i.e., $m \leq 4$. Therefore $W(P) \leq 2 \times 1/2 + 2 \times 1/4 < 5/3$. If $\sqrt{12/7} - 1 < u_2 \leq \sqrt{2} - 1$ and $(12/7)^{1/3} - 1 < u_3 \leq 2^{1/3} - 1$, then at most one more task with a utilization greater than δ but less than $(12/7)^{1/3} - 1$ can be assigned to the processor, i.e., $m \leq 4$, since $2 \times (\sqrt{12/7} - 1) + 2 \times [(12/7)^{1/3} - 1] > 1$. Therefore $W(P) \leq 2 \times 1/2 + 1/3 + 1/4 < 5/3$. If $\sqrt{12/7} - 1 < u_2 \leq \sqrt{2} - 1$ and $2^{1/3} - 1 < u_3 \leq \sqrt{12/7} - 1$, then no more task with a utilization greater than δ can be assigned to the processor, i.e., $m \leq 3$, since $2 \times (\sqrt{12/7} - 1) + 2^{1/3} - 1 + \delta > 1$. Therefore $W(P) \leq 2 \times 1/2 + 3/8 < 5/3$. If $\sqrt{12/7} - 1 < u_3 \leq u_2 \leq \sqrt{2} - 1$, then no more task with a utilization greater than δ can be assigned to the processor, i.e., $m \leq 3$, since $3 \times (\sqrt{12/7} - 1) + \delta > 1$. Therefore $W(P) \leq 3 \times 1/2 < 5/3$.

Case V: $\sqrt{2} - 1 < u_1 \leq (12/7)^{2/3} - 1$. There are several sub-cases to consider.

If $u_2 \leq (12/7)^{1/3} - 1$, then at most three more tasks each with a utilization greater than δ can be assigned to the processor, i.e., $m \leq 5$. Therefore $W(P) \leq 2/3 + 4 \times 1/4 = 5/3$.

If $(12/7)^{1/3} - 1 < u_2 \leq 2^{1/3} - 1$ and $u_3 \leq (12/7)^{1/3} - 1$, then at most one more task with a utilization greater than δ can be assigned to the processor, i.e., $m \leq 4$, since $\sqrt{2} - 1 + (12/7)^{1/3} - 1 + 3 \times \delta > 1$. Then $W(P) \leq 2/3 + 1/3 + 2 \times 1/4 < 5/3$. If $(12/7)^{1/3} - 1 < u_3 \leq u_2 \leq 2^{1/3} - 1$, then at most one more task with a utilization greater than δ but less than $(12/7)^{1/3} - 1$ can be assigned to the processor, i.e., $m \leq 4$, since $\sqrt{2} - 1 + 3 \times [(12/7)^{1/3} - 1] > 1$. Then $W(P) \leq 2/3 + 2 \times 1/3 + 1/4 < 5/3$.

If $2^{1/3} - 1 < u_2 \leq \sqrt{12/7} - 1$ and $u_3 \leq (12/7)^{1/3} - 1$, then at most one more task with a utilization greater than δ can be assigned to the processor, i.e., $m \leq 4$. Therefore $W(P) \leq 2/3 + 3/8 + 2 \times 1/4 < 5/3$. If $2^{1/3} - 1 < u_2 \leq \sqrt{12/7} - 1$ and

$(12/7)^{1/3} - 1 < u_3 \leq 2^{1/3} - 1$, then no more task with a utilization greater than δ can be assigned to the processor, i.e., $m \leq 3$. Therefore $W(P) \leq 2/3 + 3/8 + 1/3 < 5/3$. If $2^{1/3} - 1 < u_3 \leq u_2 \leq \sqrt{12/7} - 1$, then no more task with a utilization greater than δ and can be assigned to the processor, i.e., $m \leq 3$. Therefore $W(P) \leq 2/3 + 2 \times 3/8 < 5/3$.

If $\sqrt{12/7} - 1 < u_2 \leq \sqrt{2} - 1$ and $u_3 \leq 2^{1/3} - 1$, then no more task with a utilization greater than δ can be assigned to the processor, i.e., $m \leq 3$, since $\sqrt{2} - 1 + \sqrt{12/7} - 1 + 2 \times \delta > 1$. Therefore $W(P) \leq 2/3 + 1/2 + 1/3 < 5/3$. If $\sqrt{12/7} - 1 < u_2 \leq \sqrt{2} - 1$ and $2^{1/3} - 1 < u_3 \leq \sqrt{12/7} - 1$, then no more task with a utilization greater than δ can be assigned to the processor, i.e., $m \leq 3$, since $\sqrt{2} - 1 + \sqrt{12/7} - 1 + 2^{1/3} - 1 + \delta > 1$. Therefore $W(P) \leq 2/3 + 1/2 + 3/8 < 5/3$.

If $\sqrt{2} - 1 < u_2 \leq (12/7)^{2/3} - 1$, then at most one more task with a utilization greater than δ but less than $(12/7)^{1/3} - 1$ can be assigned to the processor, i.e., $m \leq 3$. Therefore $W(P) \leq 2/3 + 2/3 + 1/4 < 5/3$.

Case VI: $(12/7)^{2/3} - 1 < u_1 \leq 5/7$. There are several sub-cases to consider.

If $\delta < u_2 \leq (12/7)^{1/3} - 1$, then at most two more tasks each with a utilization greater than δ can be assigned to the processor, i.e., $m \leq 4$, since $(12/7)^{2/3} - 1 + 4 \times \delta > 1$. Therefore $W(P) \leq 3/4 + 3 \times 1/4 < 5/3$.

If $(12/7)^{1/3} - 1 < u_2 \leq 2^{1/3} - 1$ and $u_3 \leq (12/7)^{1/3} - 1$, then at most one more task with a utilization greater than δ can be assigned to the processor, i.e., $m \leq 4$, since $(12/7)^{2/3} - 1 + (12/7)^{1/3} - 1 + 3 \times \delta > 1$. Then $W(P) \leq 3/4 + 1/3 + 2 \times 1/4 < 5/3$. If $u_2 \leq 2^{1/3} - 1$ and $(12/7)^{1/3} - 1 < u_3 \leq 2^{1/3} - 1$, then at most one more task with a utilization greater than δ but less than $(12/7)^{1/3} - 1$ can be assigned to the processor, i.e., $m \leq 4$, since $(12/7)^{2/3} - 1 + 3 \times [(12/7)^{1/3} - 1] > 1$. Then $W(P) \leq 3/4 + 2/3 + 1/4 = 5/3$.

If $2^{1/3} - 1 < u_2 \leq \sqrt{12/7} - 1$ and $u_3 \leq (12/7)^{1/3} - 1$, then at most one more task with a utilization greater than δ can be assigned to the processor, i.e., $m \leq 4$. Therefore $W(P) \leq 3/4 + 3/8 + 2 \times 1/4 < 5/3$. If $2^{1/3} - 1 < u_2 \leq \sqrt{12/7} - 1$ and $(12/7)^{1/3} - 1 < u_3 \leq 2^{1/3} - 1$, then no more task with a utilization greater than δ can be assigned to the processor, i.e., $m \leq 3$. Therefore $W(P) \leq 3/4 + 3/8 + 1/3 < 5/3$. If $2^{1/3} - 1 < u_3 \leq u_2 \leq \sqrt{12/7} - 1$, then no more task with a utilization greater than δ and

can be assigned to the processor, i.e., $m \leq 3$. Therefore $W(P) \leq 3/4 + 2 \times 3/8 < 5/3$.

If $\sqrt{12/7} - 1 < u_2 \leq \sqrt{2} - 1$ and $u_3 \leq 2^{1/3} - 1$, then no more task with a utilization greater than δ can be assigned to the processor, i.e., $m \leq 3$, since $(12/7)^{2/3} - 1 + \sqrt{12/7} - 1 + 2 \times \delta > 1$, and $(12/7)^{2/3} - 1 + \sqrt{12/7} - 1 + 2^{1/3} - 1 > 1$. Therefore $W(P) \leq 3/4 + 1/2 + 1/3 < 5/3$.

If $\sqrt{2} - 1 < u_2 \leq (12/7)^{2/3} - 1$, then at most one more task with a utilization greater than δ but less than $(12/7)^{1/3} - 1$ can be assigned to the processor, i.e., $m \leq 3$. Therefore $W(P) \leq 3/4 + 2/3 + 1/4 < 5/3$.

If $(12/7)^{2/3} - 1 < u_2 \leq 5/7$, then no more task with a utilization greater than δ can be assigned to the processor, i.e., $m \leq 2$, since $2 \times [(12/7)^{2/3} - 1] + \delta > 1$. Therefore $W(P) \leq 3/4 + 3/4 < 5/3$.

Case VII: $5/7 < u_1 \leq 1$. Since $5/7 < u_1$, the total utilization of the rest of the tasks is less than $1 - 5/7 = 2/7 < \sqrt{12/7} - 1$. Furthermore, since $5/7 + 2 \times \delta > 1$, at most one more task with a utilization less than $\sqrt{12/7} - 1$ can be assigned to the processor. Therefore $W(P) \leq 1 + 3/8 < 5/3$.

Let N and N_0 be number of processors required by RM-FFDU and the minimum number of processors required to schedule a given set Σ of n tasks, respectively. Then the total weight of the task set is given by $\Sigma_{i=1}^{n} W(u_i)$. Since, except for four processors, $W(P) \geq 1$ for every processor in the RM-FFDU schedule, then $\Sigma_{i=1}^{n} W(u_i) \geq N - 4$. Since $W(P) \leq 5/3$ for every processor in the optimal schedule, $N_0 \times 5/3 \geq \Sigma_{i=1}^{n} W(u_i)$. Therefore, $\Re_A \leq 5/3$. □

Proof of Theorem 2: We first claim that $\Re_{RM-FFDU} \leq 5/3$ for $x \in (0, 0.13477]$. From Lemma 2, $\Re_{RM-FFDU} \leq 1.63$ for $x \in (0, 0.1253]$. So let us consider $x \in (0.1253, 0.13477]$. Since $0.13477 = 6(2^{1/6} - 1) - 0.6$, if a processor is assigned $n \leq 5$ tasks, then $(n+1)(2^{1/(n+1)} - 1) - x > 0.6$. If a processor is assigned $n \geq 6$ tasks, then $U \geq 6x > 0.6$. In other words, each processor in the RM-FFDU schedule has a utilization greater than 0.6. From the definition of $\Re_{RM-FFDU}$, it is clear that $\Re_{RM-FFDU} \leq 5/3$.

For $0.13477 \leq x < 0.143492 = 5(2^{1/5} - 1) - 0.6$, if a processor is assigned $n \leq 4$ tasks, then $(n+1)(2^{1/(n+1)} - 1) - x > 0.6$. If a processor is assigned $n \geq 5$ tasks, then

$U \geq 5x > 0.6$. For similar reason, $\Re_{RM-FFDU} \leq 5/3$.

What is left is to prove that $\Re_{RM-FFDU} \leq 5/3$ for $x \in [0.14349, 1/2]$. According to Lemma 3 to Lemma 8, we have $\Re_{RM-FFDU} \leq 5/3$ for $x \in [0.14349, 1/2]$. Thus we conclude that $\Re_{RM-FFDU} \leq 5/3$. □

Theorem 3 $\Re_{RM-FFDU} = 5/3$

Proof. In order to prove that the bound is tight. We need to show that the upper bounded number of processors is indeed required for some large task sets if they are scheduled by the RM-FFDU algorithm.

Let $n = 15k$, where k is a natural number.

Then we can construct a task set in term of task utilizations as follows:

$u_i = 0.2$, for $i = 1, 2, \ldots, n$.

In the completed RM-FFDU schedule, each processor is assigned three tasks since $0.2 > 2/\prod_{j=1}^{3}(1 + 0.2) - 1$. Therefore, a total of $n/3$ processors is used to schedule the task set, i.e., $N = n/3$.

In the optimal schedule, each processor is assigned exactly five tasks since $5 \times 0.2 = 1$. Hence, a total of $n/5$ processors is used to schedule the same task set, i.e., $N_0 = n/5$.

Since $N/N_0 = 5/3$, together with Theorem 2, we conclude that $\Re_{RM-FFDU} = 5/3$

□

5 The Empirical Studies of RM-FFDU

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

The simulation is conducted by running the algorithm on a large number of computer generated sample task sets and averaging the results over a number of runs. The input data of all parameters for a task set are generated according to uniform distribution. In each experiment, we vary the value of parameter α, the maximum utilization of any task in the set, i.e., $\alpha = max_i(C_i/T_i)$. The periods of tasks are generated in the range of $1 \leq T_i \leq 500$. The computation time of the tasks are taken from a range of $1 \leq C_i \leq \alpha T_i$. The output parameter for the algorithm is the percentage of extra processors used to accommodate a set of tasks, with regard to the total utilization (or load) of the task set. The total load of a task set is given by $U = \sum_{i=1}^{n} C_i/T_i$, which is a lower bound on the number of processors needed to execute the task set. In other words, the optimal number of processors needed to execute a task set with a load of U is at least U. Suppose that $N(\Sigma)$ is the number of processors required by RM-FFDU to schedule a task set Σ with a load of U, then the percentage of extra processors is defined by $100 \times \frac{N(\Sigma)-U}{U}$.

We compare the new algorithm with the existing ones in the literature in term of average-case performance. All algorithms are executed on identical task sets. The outcome of the simulation experiments is shown in Figure 1 and Figure 2. The maximum utilization of a task is set to be $\alpha = 0.5$ and $\alpha = 1$ in both sets of experiments. The number of runs for each data point is chosen to be 20, since for our experiments, 20 runs is large enough to counter the effect of "randomness". Note that our algorithm consistently outperforms those in the literature, and uses less than 30% extra processors with regard to the best solution possible. All results show that the number of processors required for each algorithm increases proportionally to the total utilization of the task set.

6 Concluding Remarks

The contributions of this paper are twofold: it proposes a new heuristic algorithm to solve an old problem and most importantly, it significantly improves the solution to the problem in term of worst-case and average-case performance. The worst-case

performance bound of the algorithm RM-FFDU is shown by intricate analysis to be 1.66..., the lowest bound ever obtained for the scheduling problem, and average-case performance of the algorithm is shown by simulation to perform consistently better than those in the literature, and use less than 30% extra processors.

Although we believe that the bound can be lowered if the UO condition in RM-FFDU is replaced by the necessary and sufficient condition, the complexity in the analysis for a lower bound can be prohibitive as we have witnessed from the above proof. Another possibility in lowering the bound further is to modify RM-FFDU to deal with the assignment of certain tasks more effective, as a number of variations of the bin-packing FFD have been designed to do. Another interesting question is to consider tasks which share resources and tasks whose deadlines do not coincide with their periods (see, e.g., [1]). These are the issues that we are currently investigating.

The rate-monotonic scheduling was first discovered around 1972-1973, and made known to the world in the seminal paper by Liu and Layland in 1973. It took about 15 years until about 1989 when rate-monotonic scheduling was used as a scheduling algorithm for a real-time operating system. Now the rate-monotonic algorithm has become widely in real-time applications. The first result on rate-monotonic scheduling heuristic for multiprocessor was derived around 1977-1978 and was presented in Dhall and Liu's 1978 paper. Interests in task scheduling on multiprocessors have rapidly increased only recently, because of the inevitable employment of multiprocessors in many real-time systems. We believe that the results presented in this paper are timely ones for the research community and for practitioners at large.

Acknowledgments: This work was supported in part by ONR, CIT, and Loral Federal Systems.

References

[1] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings. Applying New Scheduling Theory to Static Priority Preemptive Scheduling. *Software Engineering Journal*, September 1993.

[2] D. J. Brown. A Lower Bound for On-line One-dimensional Bin Packing Algorithms. Technical Report TR No. R-864, Coordinated Science Lab.,University of Illinois, Urbana, IL, 1979.

[3] A. Burchard, J. Liebeherr, Y. Oh, and S. H. Son. Real-Time Tasks to Homogeneous Multiprocessor Systems. *IEEE Transactions on Computers*, to appear.

[4] E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximate Algorithms for Bin Packing - An Updated Survey, 1985. Algorithm Design for Computer System Design, (49-106) G. Ausiello, M. Lucertinit, and P. Serafini (ed.), Springer-Verlag, NY.

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

[6] S. Davari and S. K. Dhall. On a Periodic Real-Time Task Allocation Problem. In *19th Annual Hawaii International Conference on System Sciences*, pages 133–141, 1986.

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

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

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

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* W.H. Freeman and Company, 1978.

[11] D. S. Johnson. *Near-Optimal Bin Packing Algorithms.* PhD thesis, MIT, 1973.

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

[13] M. Joseph and P. Pandya. Finding Response Times in a Real-Time System. *The Computer Journal*, 29(5):390–395, 1986.

[14] N. Karmarkar and R. M. Karp. An Efficient Approximate Scheme for the One-dimensional Bin Packing Problem. In *23rd Annual Symposium on the Foundations of Computer Science*, pages 312–320, 1982.

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

[16] 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.

[17] M. F. Liang. A Lower Bound for On-line Bin Packing. *Information Processing Letters*, 10(2):76–79, 1982.

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

[19] Y. Oh. *The Design and Analysis of Scheduling Algorithms for Real-time and Fault-tolerant Computer Systems*. PhD thesis, University of Virginia, May 1994.

[20] P. Serlin. Scheduling of Time Critical Processes. In *Spring Joint Computers Conference*, pages 925–932, 1972.

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

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

[23] W. Fernandez De La Vega and G. S. Lucker. Bin Packing can be Solved within $1 + \epsilon$ in Linear Time. *Combinatoria*, 1:349–355, 1981.

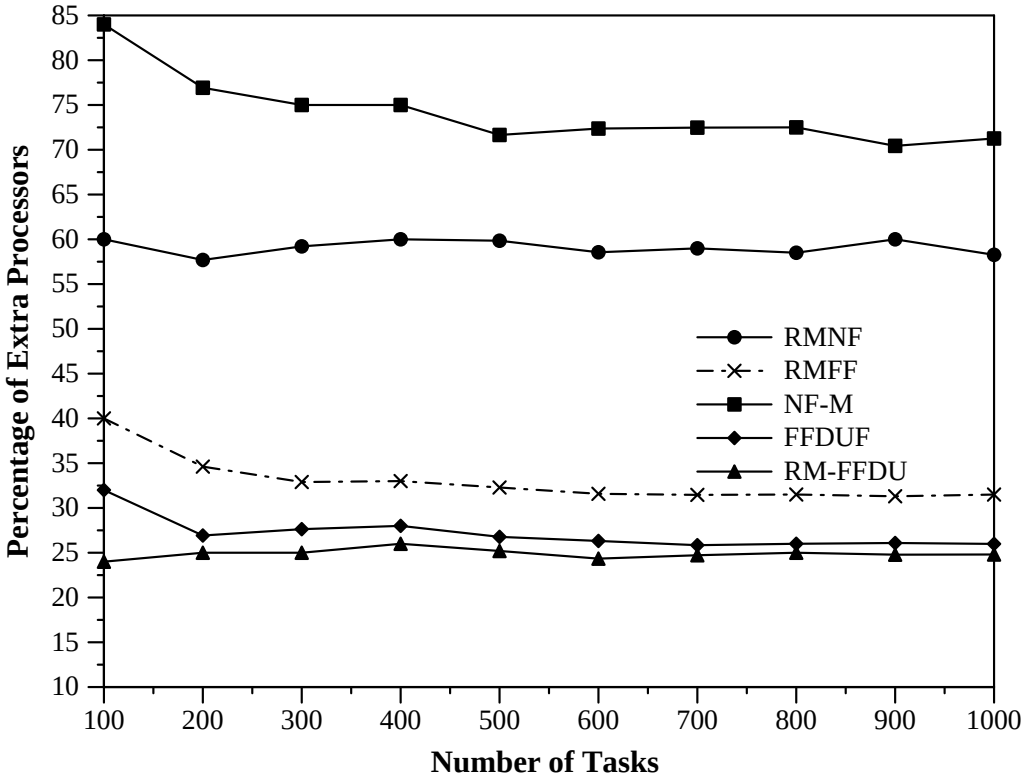

Figure 1: Performance Comparison of Algorithms $\alpha = 0.5$.

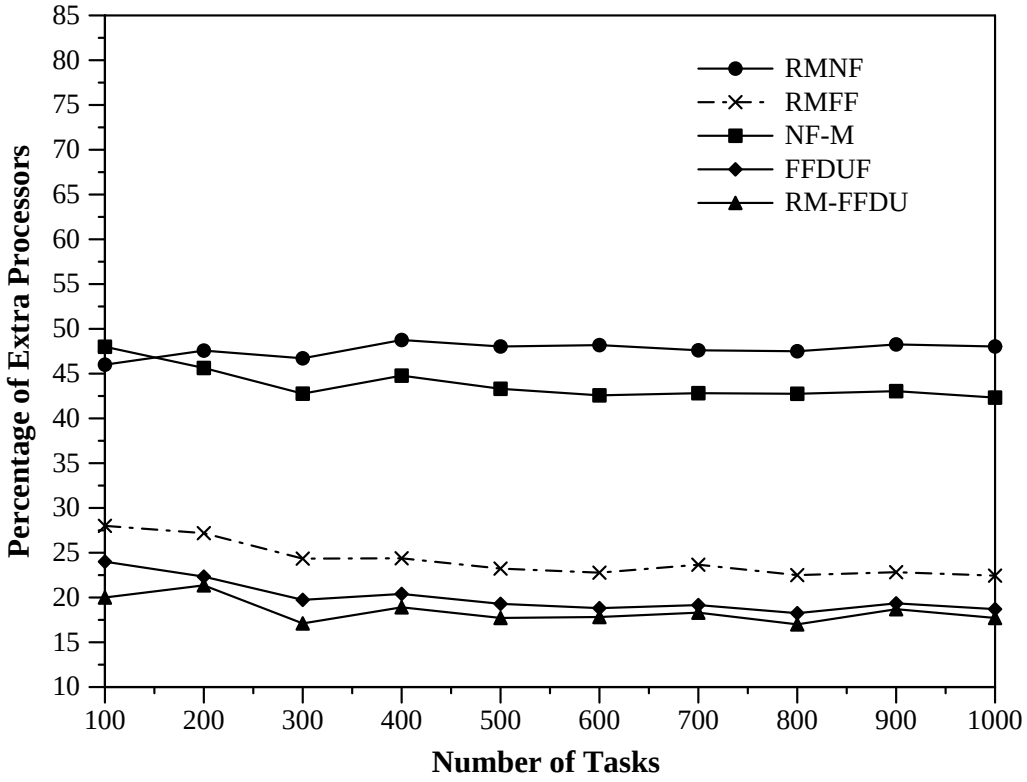

Figure 2: Performance Comparison of Algorithms $\alpha = 1.0$.