Fixed-Priority Multiprocessor Scheduling: Beyond Liu $\&$ Layland Utilization Bound

Nan Guan[1], Martin Stigge[1], Wang Yi[1] and Ge Yu[2]
[1] Uppsala University, Sweden
[2] Northeastern University, China

Abstract

The increasing interests in multicores raise the question whether utilization bounds for uni-processor scheduling can be generalized to the multiprocessor setting. Recently, this has been shown for the famous Liu and Layland utilization bound by applying novel task splitting techniques. However, parametric *utilization bounds that can guarantee higher utilizations (up to 100%) for common classes of systems are not yet known to be generalizable to multiprocessors as well. In this paper, we solve this open problem for most parametric utilization bounds by proposing new partitioning-based scheduling algorithms.*

As the second technical contribution, we show that the utilization bound proofs can be established even when exact Response Time Analysis is used for task partitioning. This enables significantly improved average-case utilization in comparison to previous work.

1 Introduction

Liu and Layland discovered the famous utilization bound $N(2^{1/N}-1)$ for fixed-priority scheduling on uni-processors in the 1970's [20]. Recently, we generalized this bound to multiprocessors by a partitioning-based scheduling algorithm [13].

The Liu and Layland utilization bound ($L\&L$ bound for short) is pessimistic: There are a significant number of task systems that exceed the $L\&L$ bound but are indeed schedulable. System resources would be considerably under-utilized if one only relies on the $L\&L$ bound in the system design.

However, if more information about the task system is available in the design phase, it is possible to derive higher *parametric* utilization bounds regarding known task parameters. A well-known example of parametric utilization bounds is the 100% bound for *harmonic* task sets [21]: If the total utilization of a harmonic task set τ is no greater than 100%, then every task in τ can meet its deadline under RMS on a uni-processor platform. Even if the whole task system is not harmonic, one can still obtain a significantly higher bound by exploring the "harmonic chains" in the system [16]. Generally, during the system design, it is usually possible to employ higher utilization bounds with available task parameter information, to better utilize the resources and decrease the system cost. As will be introduced in Section 3, quite a few higher parametric utilization bounds regarding different task parameter information have been derived for uni-processor scheduling.

This naturally raises an interesting question: Can we generalize these higher parametric utilization bounds derived for uni-processor scheduling to multiprocessors? For example, given a harmonic task system, can we guarantee the schedulability of the task system on a multiprocessor platform of M processors, if the utilization sum of all tasks in the system is no larger than M?

In this paper, we will address the above question by proposing new RMS-based partitioned scheduling algorithms (with task splitting). We first present an algorithm RM-TS/light generalizing all known parametric utilization bounds for RMS to multiprocessors, for a subclass of "light" task sets in which each task's individual utilization is at most $\frac{\Theta(\tau)}{1+\Theta(\tau)}$, where $\Theta(\tau)$ denotes the $L\&L$ bound for task set τ. Then we present the second algorithm RM-TS that works for any task set, if the parametric utilization bound is under the threshold $\frac{2\Theta(\tau)}{1+\Theta(\tau)}$[1].

Generalizing the parametric utilization bounds from uni-processors to multiprocessors is challenging, even with the insights from our previous work generalizing the $L\&L$ bound to multiprocessor scheduling. The reason is that task splitting[2] may "create" new tasks that do not comply with the parameter properties of the original task set, and thus invalidate the parametric utilization bound specific to the original task set's parameter properties. Section 3 will discuss this problem in detail. In this paper, we use more sophisticated proof techniques to solve this problem, and thereby, generalize the parametric utilization bounds to multiprocessors.

Besides the improved utilization bounds, another advantage of our new algorithms is the significantly improved average-case performance. Although the algorithm in [13] can achieve the $L\&L$ bound, it has the problem that it never utilizes more than the worst-case bound. The new algorithms in this paper use exact analysis, i.e., Response Time Analysis (RTA), instead of the utilization bound threshold as in the algorithm of [13], to determine the maximal workload on each processor. It is well-known that on uni-processors, by exact schedulability analysis, the average breakdown utilization of RMS is around 88% [19], which is much higher than its worst-case utilization bound 69.3%. Similarly, our new algorithm has much better performance than the algorithm in [13]. Due to the flexible partitioning strategy using RTA, it is a challenging problem to prove the expected utilization bounds, since processors may end up with different utilizations with the new partitioning algorithms. This breaks down the key property, required by the proof in [13], that each processor has exactly the same assigned utilization.

Related Work Multiprocessor scheduling is usually categorized into two paradigms [10]: *global scheduling*, where each task can execute on any available processor at run time, and *partitioned scheduling*, where each task is assigned to a processor beforehand, and at run time each task only executes on this fixed processor. Global scheduling on average utilizes the resources better. However, the standard RMS and EDF global scheduling strategies suffer from the notorious Dhall effect [12], which may cause a task system with arbitrarily low utilization to be non-schedulable. Although the Dhall effect can be mitigated by, e.g., assigning higher priorities to tasks with higher utilizations as in RM-US [3], the best known utilization bound of global scheduling is still quite low: 38% for fixed-priority scheduling [2] and 50% for EDF-based scheduling [7]. On the other hand, partitioned scheduling suffers from the

[1]Note that when $\Theta(\tau) \doteq 69.3\%$, $\frac{\Theta(\tau)}{1+\Theta(\tau)} \doteq 40.9\%$ and $\frac{2\Theta(\tau)}{1+\Theta(\tau)} \doteq 81.8\%$

[2]Task splitting is needed to exceed the 50% utilization bound limitation of conventional partitioned scheduling. Section 2 will introduce task splitting in detail.

resource waste similar to the bin-packing problem: A set of $M+1$ tasks of individual utilization $0.5+\epsilon$ cannot be packed in M processors, and it follows that the worst-case utilization bound for any partitioned scheduling can not exceed 50%.

Recently, a number of works [1, 4, 5, 6, 13, 14, 15, 17] have focused on partitioned scheduling with *task splitting*, which can exceed the 50% limit of the strict partitioned scheduling. In this class of scheduling, while most tasks are assigned to a fixed processor, some tasks may be (sequentially) divided into several parts and each part is assigned and thereby executed on a different (but fixed) processor. Notably, our recent work [13] has achieved the $L\&L$ bound by fixed-priority partitioned scheduling with task splitting.

2 Basic Concepts

We consider a multiprocessor platform consisting of M processors $\mathcal{P}=\{P_1,P_2,...P_M\}$. A task set $\tau=\{\tau_1,\tau_2,...,\tau_N\}$ complies with the $L\&L$ task model: Each task τ_i is a 2-tuple $\langle C_i,T_i\rangle$, where C_i is the worst-case execution time and T_i is the minimal inter-release separation (also called period). T_i is also τ_i's relative deadline. We use the RMS strategy to assign priorities: tasks with shorter periods have higher priorities. Without loss of generality we sort tasks in non-decreasing period order, and can therefore use the task indices to represent task priorities, i.e., $i<j$ implies that τ_i has higher priority than τ_j. The *utilization* of each task τ_i is defined as $U_i=C_i/T_i$, and the *total utilization* of task set τ is $\mathcal{U}(\tau)=\sum_{i=1}^{N}U_i$. We further define the *normalized utilization* of a task set τ on a multiprocessor platform with M processors:

$$\mathcal{U}_M(\tau)=\sum_{\tau_i\in\tau}U_i/M$$

Note that the subscript M in $\mathcal{U}_M(\tau)$ reminds us that the sum of all tasks' utilizations is divided by the number of processors M.

A partitioned scheduling algorithm (with task splitting) consists of two parts: the *partitioning algorithm*, which determines how to split and assign each task (or rather each of its parts) to a fixed processor, and the *scheduling algorithm*, which determines how to schedule the tasks assigned to each processor at run time.

With the partitioning algorithm, most tasks are assigned to a processor (and thereby will only execute on this processor at run time). We call these tasks *non-split tasks*. The other tasks are called *split tasks*, since they are split into several *subtasks*. Each subtask of a split task τ_i is assigned to (and thereby executes on) a different processor, and the sum of the execution times of all subtasks equals C_i. For example, in Figure 1 task τ_i is split into three subtasks τ_i^1, τ_i^2 and τ_i^3, executing on processor P_1, P_2 and P_3, respectively.

The subtasks of a task need to be synchronized to execute correctly. For example, in Figure 1, τ_i^2 should not start execution until τ_i^1 is finished. This equals deferring the actual ready time of τ_i^2 by up to R_i^1 (relative to τ_i's original release time), where R_i^1 is τ_i^1's worst-case response time. One can regard this as shortening the actual relative deadline of τ_i^2 by up to R_i^1. Similarly, the actual ready time of τ_i^3 is deferred by up to $R_i^1+R_i^2$, and τ_i^3's actual relative deadline is shortened by up to $R_i^1+R_i^2$. We use τ_i^k to denote the k^{th} subtask of a split task τ_i, and define τ_i^k's *synthetic deadline* as

$$\Delta_i^k=T_i-\sum_{l\in[1,k-1]}R_i^l. \tag{1}$$

Thus, we represent each subtask τ_i^k by a 3-tuple $\langle C_i^k,T_i,\Delta_i^k\rangle$, in which C_i^k is the execution time of τ_i^k, T_i is the original period and Δ_i^k is the synthetic deadline. For consistency, each non-split task

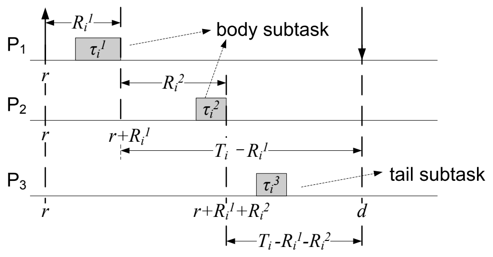

Figure 1. An Illustration of Task Splitting.

τ_i can be represented by a single subtask τ_i^1 with $C_i^1=C_i$ and $\Delta_i^1=T_i$. We use $U_i^k=C_i^k/T_i$ to denote a subtask τ_i^k's utilization.

We call the last subtask of τ_i its *tail subtask*, denoted by τ_i^t and the other subtasks its *body subtasks*, as shown in Figure 1. We use $\tau_i^{b_j}$ to denote the j^{th} body subtask.

We use $\tau(P_q)$ to denote the set of tasks τ_i assigned to processor P_q, and say P_q is the *host processor* of τ_i. We use $\mathcal{U}(P_q)$ to denote the sum of the utilization of all tasks in $\tau(P_q)$:

$$\mathcal{U}(P_q)=\sum_{\tau_i\in\tau(P_q)}U_i$$

A task set τ is *schedulable* under a partitioned scheduling algorithm $\mathcal{A}$, if (i) each task (subtask) has been assigned to some processor by $\mathcal{A}$'s partitioning algorithm, and (ii) each task (subtask) is guaranteed to meet its deadline under $\mathcal{A}$'s scheduling algorithm.

3 Deflatable Parametric Utilization Bounds

A *Parametric Utilization Bound* (PUB for short) $\Omega(\tau)$ for a task set τ is the result of applying a function $\Omega(\cdot)$ to τ's task parameters, such that all tasks in τ are guaranteed to meet their deadlines under RMS on a uni-processor if τ's total utilization $\mathcal{U}(\tau)\leq\Omega(\tau)$.

There have been quite a few parametric utilization bounds derived for RMS on uni-processors. The following are some examples:

- The famous $L\&L$ bound, denoted by $\Theta(\tau)$, is a PUB regarding the number of tasks N: $\Theta(\tau)=N(2^{1/N}-1)$
- The harmonic chain bound: HC-Bound$(\tau)=K(2^{1/K}-1)$ [16] , where K is the number of harmonic chains in the task set. The 100% bound for harmonic task sets is a special case of the harmonic chain bound with $K=1$.
- T-Bound(τ) [18] is a PUB regarding the number of tasks and the task periods: T-Bound$(\tau)=\sum_{i=1}^{N}\frac{T'_{i+1}}{T'_i}+2\cdot\frac{T'_1}{T'_N}-N$, where T'_i is τ_i's *scaled period* [18].
- R-Bound(τ) [18] is similar to T-Bound(τ), but uses a more abstract parameter r, the ratio between the minimum and maximum scaled period of the task set: R-Bound$(\tau)=(N-1)(r^{1/(N-1)}-1)+2/r-1$.

We observe that all the above PUBs have the following property: Suppose a PUB $\Omega(\tau)$ is derived from a task set τ's parameters. If we decrease the execution times of some tasks in τ to get a new task set τ', then $\Omega(\tau)$ is still applicable to τ'. We call a PUB holding this property a *deflatable* parametric utilization bound, as formally stated in the following definition:

Definition 1. *A* Deflatable *Parametric Utilization Bound (*D-PUB*) $\Omega(\tau)$ is a* PUB *satisfying the following property: We decrease the execution times of some tasks in τ to get a new task set τ'. If τ' satisfies $\mathcal{U}(\tau')\leq\Omega(\tau)$, then it is guaranteed to be schedulable by* RMS *on a uni-processor.*

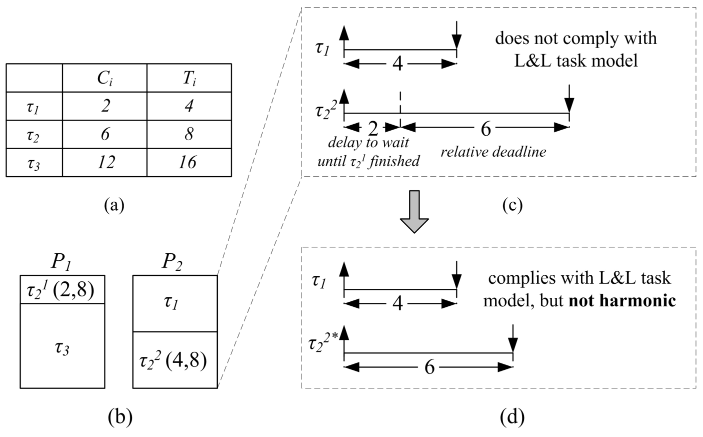

Figure 2. Partitioning a harmonic task set results in a nonharmonic task set on some processor.

We would like to point out that the deflatable property is different from the (self-)sustainable property [8] [9]. The deflatable property does *not* require the original task set τ to satisfy $\mathcal{U}(\tau) \leq \Omega(\tau)$ (τ will be scheduled on M processors and in general have total utilization lager than 100%). $\Omega(\tau)$ is merely a value obtained by applying the function $\Omega(\cdot)$ to τ's parameters, and will be used as a utilization bound to each individual processor.

The *deflatable* property is very common for PUBs: In fact all PUBs for RMS on uni-processors we are aware of are deflatable[3]. In the following, we use $\Omega(\tau)$ to denote an arbitrary D-PUB derived from τ's parameters under RMS on uni-processors.

D-PUBs are of great relevance to partitioned multiprocessor scheduling, since a task set τ will be partitioned into several subsets and each subset is executed on a processor individually. Further, due to the task splitting, a task could be divided into several subtasks, each of which holds a portion of the execution demand of the original task. So the D-PUB property is clearly useful to generalize a utilization bound to multiprocessors.

However, the deflatable property by itself is *not sufficient* for the generalization of $\Omega(\tau)$ to multiprocessors. For example, suppose the harmonic task set τ in Figure 2-(a) is partitioned as in Figure 2-(b), where τ_2 is split into τ_2^1 and τ_2^2. On P_2, to correctly execute τ_2, τ_2^1 and τ_2^2 need to be synchronized such that τ_2^2 never starts execution before its precedence τ_2^1 is finished. This can be viewed as shortening τ_2^2's relative deadline for a certain amount of time from τ_2's original deadline, as shown in Figure 2-(c). In this case, τ_2^2 does not comply with the $L\&L$ task model (which requires the relative deadline to equal the period), so none of the parametric utilization bounds for the $L\&L$ task model is applicable to processor P_2. In [13], this problem is solved by representing τ_2^2's period by its relative deadline, as shown in Figure 2-(d). This transforms the task set $\{\tau_1, \tau_2^2\}$ into a $L\&L$ task set $\{\tau_1, \tau_2^{2*}\}$, with which we can apply the $L\&L$ bound. However, this solution does not in general work for other parametric utilization bounds: In our example, we still want to apply the 100% bound which is specific to harmonic task sets. But if we use τ_2^2's deadline 6 to represent its period 8, the task set $\{\tau_1, \tau_2^{2*}\}$ is not harmonic, so the 100% bound is not applicable. This problem will be solved by our new algorithms and novel proof techniques in the following sections.

[3]The PUBs we are aware of include the ones listed above, and the non-closed-form bounds in [11]. We do not exclude the possibility that there might exist (undiscovered) parametric utilization bounds that are *not* deflatable. However, proving the existence of, or finding such a non-deflatable bound is out of the scope of this paper.

4 The Algorithm for Light Tasks: RM-TS/light

In the following we introduce the first algorithm RM-TS/light, which achieves $\Omega(\tau)$ (any D-PUB derived from τ's parameters), if τ is *light* in the sense of an upper bound on each task's individual utilization as follows.

Definition 2. *A task τ_i is a* light task *if*

$$U_i \leq \frac{\Theta(\tau)}{1+\Theta(\tau)} \tag{2}$$

where $\Theta(\tau)$ denotes the L&L bound. Otherwise, τ_i is a heavy task. *A task set τ is a* light task set *if all tasks in τ are light tasks.*

RM-TS/light follows two simple principles: (1) assigning tasks to processors in increasing priority order, and (2) packing tasks to processors by a "width-first" strategy such that the utilizations on all processors are increased "evenly". In this way, when task splitting occurs, most capacity of the processors has been occupied by tasks with relatively low priorities, and the unassigned tasks all have relatively high priorities. This is good for a higher accepted utilization on each processor: After being split, a subtask τ_i^k's actual deadline Δ_i^k is shorter than τ_i's original deadline (period) T_i. In fixed-priority scheduling, it is generally the interference suffered by *low-priority* tasks that determines the maximal accepted utilization. Thus, in general, the total utilization accepted by a processor will decrease if one shortens the deadline of a low-priority task. In contrast, if one shortens the deadline of a high-priority task, there is a better chance that the accepted utilization is not affected.

The reason for restricting RM-TS/light to light task sets is that, a task with very high utilization may need to be split even when the occupied utilizations of all processors are relatively low. The second algorithm presented in Section 5 will work on task sets that also include *heavy* tasks.

4.1 Algorithm Description

The partitioning algorithm of RM-TS/light is quite simple. We describe it briefly as follows:

- Tasks are assigned in increasing priority order. We always select the processor on which the total utilization of the tasks that have been assigned so far is *minimal* among all processors.
- A task (subtask) can be entirely assigned to the current processor, if all tasks including this one on this processor can meet their deadlines under RMS.
- When a task (subtask) cannot be assigned entirely to the current processor, we split it into two parts[4]. The first part is assigned to the current processor. The splitting is done such that the portion of the first part is as big as possible, guaranteeing no task on this processor misses its deadline under RMS; the second part is left for the assignment to the next selected processor.

In the following, we will give a detailed description. Algorithm 1 and 2 describe the partitioning algorithm of RM-TS/light in pseudo-code. At the beginning, tasks are sorted (and will therefore be assigned) in increasing priority order, and all processors are marked as *non-full* which means they still can accept more tasks. At each step, we pick the next task in order (the one with the lowest priority), select the processor with the minimal total utilization of

[4]In general a task may be split into more than two subtasks. Here we mean at each step the currently selected task (subtask) is split into two parts.

tasks that have been assigned so far, and invoke the routine Assign to do the task assignment. Assign first verifies that after assigning the task, all tasks on that processor would still be schedulable under RMS. This is done by applying exact schedulability analysis of calculating the response time R_j^k of each task τ_j^k after assigning the new task τ_i^k to P_q with the well-known fixed-point formula:

$$R_j^k = \sum_{\substack{\tau_h \in \tau(P_q) \\ h<j}} \left\lceil \frac{R_j^k}{T_h} \right\rceil C_h + C_j^k$$

The response time R_j^k obtained for each (sub)task τ_j^k is compared to its (synthetic) deadline Δ_j^k. If the response time does not exceed the synthetic deadline for any of the tasks on P_q, we can conclude that τ_i^k can safely be assigned to P_q without causing any deadline miss. Note that a subtask's synthetic deadline Δ_j^k may be different from its period T_j. After presenting how the overall partitioning algorithm works, we will show how to calculate Δ_j^k easily.

```
1: Task order $\tau_N^1, \ldots, \tau_1^1$ by increasing priorities
2: Mark all processors as *non-full*
3: **while** there is an *non-full* processor **and** an unassigned task **do**
4:     Pick next task $\tau_i^k$,
5:     Pick *non-full* processor $P_q$ with minimal $\mathcal{U}(P_q)$
6:     Assign$(\tau_i^k, P_q)$
7: **end while**
8: If there is an unassigned task, the algorithm **fails**, otherwise it **succeeds**.
```

Algorithm 1: The partitioning algorithm of RM-TS/light.

```
1: **if** $\tau(P_q)$ with $\tau_i^k$ is still schedulable **then**
2:     Add $\tau_i^k$ to $\tau(P_q)$
3: **else**
4:     Split $\tau_i^k$ via $(\tau_i^k, \tau_i^{k+1}) :=$ MaxSplit$(\tau_i^k, P_q)$
5:     Add $\tau_i^k$ to $\tau(P_q)$
6:     Mark $P_q$ as *full*
7:     $\tau_i^{k+1}$ is next task
8: **end if**
```

Algorithm 2: The Assign(τ_i^k, P_q) routine.

If τ_i^k cannot be entirely assigned to the currently selected processor P_q, it will be split into two parts using routine MaxSplit(τ_i^k, P_q): one subtask that makes maximum use of the selected processor, and a remaining part of that task, which will be subject to assignment in the next iteration. The desired property here is that we want the first part to be as big as possible such that, after assigning it to P_q, all tasks on that processor will still be able to meet their deadlines. In order to state the effect of MaxSplit formally, we introduce the concept of a *bottleneck*.

Definition 3. *A* bottleneck *of processor P_q is a (sub)task that is assigned to P_q, and will become non-schedulable if we increase the execution time of the task with the highest priority on P_q by an arbitrarily small positive number.*

Note that there may be more than one bottleneck on a processor. Further, since RM-TS/light assigns tasks in increasing priority order, MaxSplit always operates on the task that has the highest priority on the processor in question. Thus, we can state:

Definition 4. MaxSplit(τ_i^k, P_q) *is a function that splits τ_i^k into two subtasks τ_i^k and τ_i^{k+1} such that:*

1. *τ_i^k can now be assigned to P_q without making any task in $\tau(P_q)$ non-schedulable.*
2. *After assigning τ_i^k, P_q has a bottleneck.*

MaxSplit can be implemented by, for example, performing a binary search over $[0, C_i^k]$ to find out the maximal portion of τ_i^k with which all tasks on P_q can meet their deadlines. A more efficient implementation of MaxSplit was presented in [17], in which one only needs to check a (small) number of possible values in $[0, C_i^k]$. The complexity of this improved implementation is still pseudo-polynomial, but in practise it is very efficient.

The while loop in RM-TS/light terminates as soon as all processors are "full" *or* all tasks have been assigned. If the loop terminates due to the first reason and there are still unassigned tasks left, the algorithm reports a failure of the partitioning, otherwise a success.

Calculating Synthetic Deadlines Now we will show how to calculate each (sub)task τ_i^k's synthetic deadline Δ_i^k, which was left open in the above presentation. If τ_i^k is a non-split task, its synthetic deadline trivially equals its period T_i. Now we consider the case that τ_i^k is a subtask of a split task τ_i. Recall that tasks are assigned in increasing order of priorities. Thus, right after a (sub)task is split and assigned to its host processor, the first part of it, which is a body subtask, has the highest priority on that processor. After that the processor will be marked as *full* and consequently no other tasks of higher priority can be assigned to it. So we know:

Lemma 1. *A body subtask has the highest priority on its host processor.*

A consequence of this is, the response time of each body subtask equals its execution time, and one can replace R_i^l by C_i^l in (1) to calculate the synthetic deadline of a subtask. Especially, we are interested in the synthetic deadlines of tail subtasks (we don't need to worry about a body subtask's synthetic deadline since it has the highest priority on its host processor and is schedulable anyway). The calculation is explicitly stated in the following lemma.

Lemma 2. *Let τ_i be a task split into B_i body subtasks $\tau_i^{b_1}, \ldots, \tau_i^{b_{B_i}}$, assigned to processors $P_{b_1}, \ldots, P_{b_{B_i}}$ respectively, and the tail subtask τ_i^t assigned to processor P_t. The synthetic deadline Δ_i^t of a tail subtask τ_i^t is calculated by:*

$$\Delta_i^t = T_i - \sum_{j \in [1, B_i]} C_i^{b_j}$$

Scheduling at Run Time At runtime, the tasks will be scheduled using RMS on each processor locally, i.e., with their original priorities. The subtasks of a split task respect their precedence relations, i.e., a split subtask τ_i^k is ready for execution when its preceding subtask τ_i^{k-1} on some other processor has finished.

From the presented partitioning and scheduling algorithm of RM-TS/light, it is clear that successful partitioning implies schedulability, i.e., the guarantee that all deadlines can be met.

Lemma 3. *Any task set that has been successfully partitioned by* RM-TS/light *is schedulable.*

4.2 Utilization Bound

We will now prove that RM-TS/light has the utilization bound of $\Omega(\tau)$ for *light* task sets. First we will briefly introduce the main idea and structure of the proof.

To show that RM-TS/light has a utilization bound $\Omega(\tau)$, we will prove that if a light task set τ is non-schedulable, i.e., not successfully partitioned by RM-TS/light, then the sum of the assigned utilizations of all processors is *at least*[5] $M \cdot \Omega(\tau)$.

[5]By this, the normalized utilization of τ *strictly exceeds* $\Omega(\tau)$, since there are (sub)tasks not assigned to any of the processors after a failed partitioning.

In order to show this, we assume that the assigned utilization on some processor is *strictly less* than $\Omega(\tau)$. We prove that this implies there is no bottleneck on that processor. This is a contradiction, because each processor on which MaxSplit has been used must have a bottleneck afterwards. We also know that MaxSplit was used for all processors, since the partitioning failed.

In the following, we assume P_q to be a processor with an assigned utilization of $U(P_q) < \Omega(\tau)$. A task on P_q is either a non-split task, a body subtask or a tail subtask. The main part of the proof consists of showing that P_q cannot have a bottleneck of any type.

As the first step, we show this for non-split tasks and body subtasks (Lemma 4), after which we turn to the more difficult case of tail subtasks (Lemma 6).

Lemma 4. *Suppose task set τ is not schedulable by* RM-TS/light*, and after the partitioning phase it holds for a processor P_q that*

$$\mathcal{U}(P_q) < \Omega(\tau)$$

Then a bottleneck of P_q is neither a non-split task nor a body subtask.

Proof. By Lemma 1 we know that the body subtask has the highest priority on P_q, so it can never be a bottleneck.

For the case of non-split tasks, note that all tasks are scheduled locally by RMS using their original periods, and a non-split task's parameter (particularly its deadline) is unchanged. We use Γ to denote the set of tasks on P_q, and construct a new task set Γ^* corresponding to Γ such that each non-split task τ_i in Γ has a counterpart in Γ^* that is exactly the same as τ_i, and each subtask in Γ has a counterpart in Γ^* with deadline changed to equal its period. Γ^* complies with the $L\&L$ task model, and can be viewed as obtained from the whole task set τ by decreasing some of the tasks' execution times (some tasks' execution times are decreased to 0, which equals that these tasks are eliminated). So the D-PUB $\Omega(\tau)$ is sufficient to the guarantee the schedulability of Γ^*. Thus, if the execution time of the highest-priority task on P_q is increased by an (arbitrarily small) amount ε such that the total utilization still does not exceed $\Omega(\tau)$, Γ^* will still be schedulable. Recall that the only difference between Γ and Γ^* is the subtasks' deadlines, and since the scheduling of RMS does depend on tasks' deadline, we can conclude that each non-split task in Γ is also schedulable (after increasing ε to the highest priority task on P_q). Note that for this moment we only want to guarantee the schedulability of non-split tasks in Γ, but do not care whether the tail subtasks in Γ can meet deadlines or not. □

After the cases of non-split tasks and body subtasks, now we prove that in a light task set, a bottleneck on a processor with utilization lower than $\Omega(\tau)$ is not a tail subtask either. The proof goes in two steps: We first derive in Lemma 5 a general condition guaranteeing that a tail subtask can not be a bottleneck; then we conclude in Lemma 6 that a bottleneck on a processor with utilization lower than $\Omega(\tau)$ is not a tail subtask, by showing that the condition in Lemma 5 holds for each of these tail subtasks.

We use the following notation: Let τ_i be a task split into B_i body subtasks $\tau_i^{b_1}, \tau_i^{b_2}, ..., \tau_i^{b_{B_i}}$, assigned to processors $P_{b_1}, P_{b_2}, ..., P_{b_{B_i}}$ respectively, and a tail subtask τ_i^t assigned to processor P_t. The utilization of the tail subtask τ_i^t is $U_i^t = \frac{C_i^t}{T_i}$, and the utilization of a body subtask $\tau_i^{b_j}$ is $U_i^{b_j} = \frac{C_i^{b_j}}{T_i}$. We use U_i^{body} to denote the total utilization of τ_i's all body subtasks, i.e.,

$$U_i^{body} = \sum_{j \in [1, B_i]} U_i^{b_j} = U_i - U_i^t$$

For each body subtask $\tau_i^{b_j}$, let X_{b_j} denote the total utilization of all (sub)tasks assigned to P_{b_j} with *lower* priority than $\tau_i^{b_j}$. For the tail subtask τ_i^t, let X_t denote the total utilization of all (sub)tasks assigned to P_t with *lower* priority than τ_i^t, and Y_t the total utilization of all (sub)tasks assigned to P_t with *higher* priority than τ_i^t.

Now we start with the general condition identifying non-bottleneck tail subtasks.

Lemma 5. *Suppose a tail subtask τ_i^t is assigned to processor P_t and $\Theta(\tau)$ is the L&L bound. If it holds that*

$$Y_t + U_i^t < \Theta(\tau) \cdot (1 - U_i^{body}) \qquad (3)$$

then τ_i^t is not *a bottleneck of processor P_t.*

Proof (Sketch). We only introduce the brief idea of the proof due to space limit.

We increase the utilization of the task with the highest priority on P_t by a small number ϵ such that it holds (one can always find such an ϵ):

$$(Y_t + \epsilon) + U_i^t < \Theta(\tau) \cdot (1 - U_i^{body})$$

By the definition of U_i^{body} and Δ_i^t, this equals

$$((Y_t + \epsilon) + U_i^t) \cdot T_i / \Delta_i^t < \Theta(\tau) \qquad (4)$$

The key of the proof is to show that Condition (4) still guarantees that τ_i^t can meet its deadline. We consider the task set Γ consisting of τ_i^t and all tasks with higher priorities on P_t. Then we construct a task set Γ^* corresponding to Γ, such that each task in Γ has a counterpart in Γ^* but with a period reduced to Δ_i^t in case its original period exceeds Δ_i^t, and then a deadline changed to equal its period in case they are different. The constructed task set Γ^* complies with the Liu and Layland task model (deadlines equal periods) and is prioritized by periods. Thus, one can apply the $L\&L$ bound $\Theta(\tau)$ to it. At the same time, the total utilization of Γ^* is bounded by $((Y_t + \epsilon) + U_i^t)T_i/\Delta_i^t$, so Condition (4) implies the counterpart of τ_i^t can meet its deadline in the context of Γ^* (note that here we only focus on the schedulability of τ_i^t, and the schedulability of other tail subtasks are guaranteed when this lemma is applied to themselves). Further, since Γ^* is constructed in a way that each task's workload is at most that of its counterpart in Γ^*, the interference τ_i^t suffered in Γ is at most the interference in Γ^*. From this we know that τ_i^t can meet its deadline in the context of Γ, which establishes the proof. □

Note that in the above proof we use the $L\&L$ bound $\Theta(\tau)$ rather than the higher bound $\Omega(\tau)$ to guarantee the schedulability of tasks in Γ^*, since the task periods are modified in the construction of Γ^*, and $\Omega(\tau)$ may not apply to Γ^* (recall that the *deflatable* property of $\Omega(\tau)$ only tolerates changes to execution times, but not periods). For example, suppose the original task set is harmonic, the constructed set Γ^* may not be harmonic since some of task periods are shorten to Δ_i^t, which is not necessarily harmonic with other periods. So the 100% bound of harmonic task sets does not apply to Γ^*. However, $\Theta(\tau)$ is still applicable, since it only depends on, and is monotonically decreasing regarding the number of tasks.

Having this lemma, we now show that a tail subtask τ_i^t cannot be a bottleneck either, if its host processor's utilization is less than $\Omega(\tau)$, by proving Condition (3) for τ_i^t.

Lemma 6. *Let τ be a* light *task set non-schedulable by* RM-TS/light*, and let τ_i be a split task whose tail subtask τ_i^t is assigned to processor P_t. If*

$$\mathcal{U}(P_t) < \Omega(\tau) \qquad (5)$$

then τ_i^t is not a bottleneck of P_t.

Proof. The proof is by contradiction. We assume the lemma does *not* hold for one or more tasks, and let τ_i be the lowest-priority one among these tasks, i.e., τ_i^t is a bottleneck of its host processor P_t, and all tail subtasks with lower priorities are either not a bottleneck or on a processor with assigned utilization at least $\Omega(\tau)$.

Recall that $\{\tau_i^{b_j}\}_{j \in [1, B_i]}$ are the B_i body subtasks of τ_i, and $\{P_{b_j}\}_{j \in [1, B_i]}$ and P_t are processors hosting the corresponding body and tail subtasks. Since a body task has the highest priority on its host processor (Lemma 2) and tasks are assigned in increasing priority order, all tail subtasks on processors $\{P_{b_j}\}_{j \in [1, B_i]}$ have lower priorities than τ_i.

We will first show that all processors $\{P_{b_j}\}_{j \in [1, B_i]}$ have an individual assigned utilization at least $\Omega(\tau)$. We do this by contradiction. Assume there is a P_{b_j} whose assigned utilization is lower than $\Omega(\tau)$. By above discussions we know that the tail subtasks on P_{b_j} cannot be bottlenecks, and by Lemma 4 we know that a bottleneck of P_{b_j} is neither a non-split task nor a body subtask. This implies there is no bottleneck on P_{b_j} which contradicts the fact there is at least one bottleneck on each processor. So the assumption of P_{b_j}'s assigned utilization being lower than $\Omega(\tau)$ must be false, by which we can conclude that all processors hosting τ_i^t's body tasks have assigned utilization at least $\Omega(\tau)$. Thus we have:

$$\sum_{j\in[1,B_i]} \underbrace{(U_i^{b_j} + X_{b_j})}_{\mathcal{U}(P_{b_j})} \geq B_i \cdot \Omega(\tau) \qquad (6)$$

Further, the assumption from Condition (5) can be rewritten as:

$$X_t + Y_t + U_i^t < \Omega(\tau) \qquad (7)$$

We combine (6) and (7) into:

$$X_t + Y_t + U_i^t < \frac{1}{B_i} \sum_{j\in[1,B_i]} (U_i^{b_j} + X_{b_j})$$

Since the partitioning algorithm selects at each step the processor on which the so-far assigned utilization is minimal, we have $\forall j \in [1, B_i] : X_{b_j} \leq X_t$. Thus, the inequality can be relaxed to:

$$Y_t + U_i^t < \frac{1}{B_i} \sum_{j\in[1,B_i]} U_i^{b_j}$$

We also have $B_i \geq 1$ and $U_i^{body} = \sum_{j\in[1,B_i]} U_i^{b_j}$, so:

$$Y_t + U_i^t < U_i^{body}$$

Now, in order to get to Condition (3), which implies τ_i^t is not a bottleneck (Lemma 5), we need to show that the RHS of this inequality is bounded by the RHS of Condition (3), i.e., that:

$$U_i^{body} \leq \Theta(\tau)(1 - U_i^{body})$$

It is easy to see that this is equivalent to the following, which holds since τ_i is by assumption a light task:

$$U_i^{body} \leq \frac{\Theta(\tau)}{1 + \Theta(\tau)}$$

□

Now we put all parts of the proof in place and can summarize them as the main theorem concerning the utilization bound of RM-TS/light for light task sets.

Theorem 7. *$\Omega(\tau)$ is a utilization bound of* RM-TS/light *for light task sets, i.e., any light task set τ with*

$$\mathcal{U}_M(\tau) \leq \Omega(\tau)$$

is schedulable by RM-TS/light.

Proof. Assume a light task set τ with $\mathcal{U}_M(\tau) \leq \Omega(\tau)$ is not schedulable by RM-TS/light, i.e., there are tasks not assigned to any of the processors after the partitioning procedure with τ. By this we know the sum of the assigned utilization of all processors after the partitioning is *strictly less* than $M \cdot \Omega(\tau)$, so there is at least one processor P_q with a utilization *strictly less* than $\Omega(\tau)$. By Lemma 4 we know the bottleneck of this processor is neither a non-split task nor a body subtask, and by Lemma 6 we know the bottleneck is not a tail subtask either, so there is no bottleneck on this processor. This contradicts the property of the partitioning algorithm, that all processors to which no more task can be assigned must have a bottleneck (Definition 4). □

5 The Algorithm for Any Task Set: RM-TS

In this section, we introduce RM-TS, which removes the restriction to light task sets in RM-TS/light. We will show that RM-TS can achieve a D-PUB $\Omega(\tau)$ for any task set τ, if $\Omega(\tau)$ does not exceed $\frac{2\Theta(\tau)}{1+\Theta(\tau)}$. In other words, if one can derive a D-PUB $\Omega'(\tau)$ from τ's parameters under uni-processor RMS, RM-TS can achieve the utilization bound of $\Omega(\tau) = \min(\Omega'(\tau), \frac{2\Theta(\tau)}{1+\Theta(\tau)})$. Note $\frac{2\Theta(\tau)}{1+\Theta(\tau)} = 81.8\%$ when $\Theta(\tau) = 69.3\%$. So we can see that despite an upper bound on $\Omega(\tau)$, RM-TS still provides significant room for higher utilization bounds.

For simplicity of presentation, we assume that each task's utilization is bounded by $\Omega(\tau)$, so the utilization of a heavy task is in the range $\left(\frac{\Theta(\tau)}{1+\Theta(\tau)}, \Omega(\tau)\right]$. Note that this assumption does not invalidate the utilization bound of our algorithm for task sets which have some individual task's utilization above $\Omega(\tau)$[6].

As introduced in the beginning of Section 4, the reason why we assume a restriction to *light* tasks in RM-TS/light is that, a *heavy* task could already be split when the assigned utilizations of all processors are still very low. This could cause tail subtasks to end up with relatively low priorities on their host processors. To solve this problem, RM-TS adds a sophisticated heavy task pre-assignment mechanism to the partitioning algorithm. In the pre-assignment, we first identify the heavy tasks whose tail subtasks would have low priority if they were split, and pre-assign these tasks to one processor each. This avoids the split. The identification is checked by a simple test condition, called *Pre-assign Condition*. Those heavy tasks that do not satisfy this condition will be assigned (and possibly split) later, together with the light tasks.

Before the detailed description of RM-TS, we introduce some notations. If a heavy task τ_i is pre-assigned to a processor P_q in RM-TS, we call τ_i a *pre-assigned task* and P_q a *pre-assigned processor*, otherwise τ_i a *normal task* and P_q a *normal processor*.

5.1 Algorithm Description

The partitioning algorithm of RM-TS is shown in Algorithm 3, which contains three main phases:

1. We first pre-assign the heavy tasks that satisfy the *Pre-assign Condition* to one processor each, in decreasing priority order.
2. We do task partitioning with the remaining (i.e. normal) tasks and remaining (i.e. normal) processors similar to RM-TS/light until all the normal processors are full.
3. The remaining tasks are assigned to the pre-assigned processors in increasing priority order; the assignment selects the processor with the largest index (i.e., the one hosting the lowest-priority pre-assigned task), to assign as many tasks as possible until it is full, then selects the next processor.

The pseudo-code of RM-TS is given in Algorithm 3. At the beginning, all the processors are marked as *normal* and *non-full*.

In the first phase, we visit all the tasks in decreasing priority order, and for each task we use DeterminePreAssign(τ_i) (Algorithm 4) to determine whether we should pre-assign it. $\mathcal{P}^{\triangleright}(\tau_i)$ records the set of processors marked as *normal* at the moment we are chekcing for τ_i. If τ_i is a heavy task, we check the *Pre-assign Condition*:

$$\sum_{i<j} U_j \leq (|\mathcal{P}^{\triangleright}(\tau_i)| - 1) \cdot \Omega(\tau) \qquad (8)$$

$|\mathcal{P}^{\triangleright}(\tau_i)|$ is the number of elements in $\mathcal{P}^{\triangleright}(\tau_i)$, i.e., the number of processors marked as *normal* at this moment. If this condition is satisfied, we pre-assign this heavy task to the current selected processor, which is the one with the minimal index among all normal processors, and mark this processor as *pre-assigned*. Otherwise, we do not pre-assign this heavy task, and leave it to the following phases. The intuition of the pre-assign condition (8) is: We pre-assign a heavy task τ_i if the total utilization of lower-priority tasks is relatively small, since otherwise its tail subtask may end up with a low priority on the corresponding processor.

In the second phase we assign the remaining tasks to *normal* processors only. Note that the remaining tasks are all light tasks and

[6]One can let tasks with a utilization of more than $\Omega(\tau)$ execute exclusively on one dedicated processor each. If we can prove that the utilization bound of all the other tasks on all the other processors is $\Omega(\tau)$, then the utilization bound of the overall system is also at least $\Omega(\tau)$.

the heavy tasks that do not satisfy the Pre-assign Condition. The assignment policy in this phase is the same as for RM-TS/light: We sort tasks in increasing priority order, and at each step select the (normal) processor P_q with the minimal assigned utilization. Then we invoke routine Assign(τ_i^k, P_q), which is exactly the same as the one used by RM-TS/light. It either adds τ_i^k to $\tau(P_q)$ if τ_i^k can be entirely assigned to P_q, or splits τ_i^k and assigns a maximized portion of it to P_q otherwise.

```
1: Mark all processors as *normal* and *non-full*

   // Phase 1: Pre-assignment
2: Sort all tasks in $\tau$ in *decreasing* priority order
3: for each task in $\tau$ do
4:     Pick next task $\tau_i$
5:     if DeterminePreAssign($\tau_i$) then
6:         Pick the *normal* processor with the minimal index $P_q$
7:         Add $\tau_i$ to $\tau(P_q)$
8:         Mark $P_q$ as *pre-assigned*
9:     end if
10: end for

   // Phase 2: Assign remaining tasks to normal processors
11: Sort all unassigned tasks in *increasing* priority order
12: while there is a *non-full normal* processor
          and an unassigned task do
13:     Pick next unassigned task $\tau_i$
14:     Pick the *non-full normal* processor $P_q$ with minimal $\mathcal{U}(P_q)$
15:     $Assign(\tau_i^k, P_q)$
16: end while

   // Phase 3: Assign remaining tasks to pre-assigned processors
   // Remaining tasks are still in increasing priority order
17: while there is a *non-full pre-assigned* processor
          and an unassigned task do
18:     Pick next unassigned task $\tau_i$
19:     Pick the *non-full pre-assigned* processor $P_q$ with the largest index
20:     $Assign(\tau_i^k, P_q)$
21: end while

22: If there is an unassigned task, the algorithm fails, otherwise it succeeds.
```

Algorithm 3: The partitioning algorithm of RM-TS.

```
1: $\mathcal{P}^{\triangleright}(\tau_i)$ := the set of *normal* processors at this moment
2: if $\tau_i$ is *heavy* then
3:     if $\sum_{j>i} U_j \leq (|\mathcal{P}^{\triangleright}(\tau_i)| - 1) \cdot \Omega(\tau)$ then
4:         return true
5:     end if
6: end if
7: return false
```

Algorithm 4: The DeterminePreAssign(τ_i) routine.

In the third phase we continue to assign the remaining tasks to *pre-assigned* processors. There is an important difference between the second phase and the third phase: In the second phase tasks are assigned by a "width-first" strategy, i.e., the utilization of all processors are increased "evenly", while in the third phase tasks are now assigned by a "depth first" strategy. More precisely, in the third phase we select the pre-assigned processor with the largest index, which hosts the lowest-priority pre-assigned task of all non-full processors. We assign as much workload as possible to it, until it is full, and then move to the next processor. This strategy is one of the key points to facilitate the induction-based proof of the utilization bound in the next subsection.

Note that, in both the second and third phase we use the same routine Assign(τ_i^k, P_q) as in RM-TS/light (to split and assign tasks). In Assign(τ_i^k, P_q), the synthetic deadlines of tail subtasks are calculated under the assumption that all body subtasks have the highest priority on their host processors, which is true for RM-TS/light (Lemma 1). It is easy to see this property also holds for the second phase of RM-TS (the task assignment on normal processors), in which tasks are assigned in exactly the same way as RM-TS/light. But it is not clear for this moment whether this assumption also holds for the third phase or not, since there are pre-assigned tasks already assigned to these pre-assigned processors in the first phase, and there is a risk that a pre-assigned might have higher priority than the body subtask on that processor. However, as will be shown in the proof of Lemma 14, *a body subtask on a pre-assigned processor has the highest priority on its host processor*, and Assign(τ_i^k, P_q) still guarantees the schedulability of the split tasks involved in the third phase.

After these three phases, the partitioning failed if there still are unassigned tasks left, otherwise it is successful. At run-time, the tasks assigned to each processor are scheduled by RMS with their original priorities, and the subtasks of a split task need to respect their precedence relations, which is the same as in RM-TS/light. Assuming that a body subtask on a pre-assigned processor also has the highest priority among all tasks assigned to that processor, any task set successfully partitioned by RM-TS is guaranteed to be schedulable (by the algorithm construction). In Lemma 14 we will show that this assumption is true, and thereby prove that a successful partitioning indeed implies the schedulability.

5.2 Utilization Bound

We will now prove the utilization bound $\Omega(\tau)$ for RM-TS. It follows a similar pattern as the proof for RM-TS/light, by assuming a task set τ that can't be completely assigned. The main difficulty is that we now have to deal with heavy tasks as well. Recall that the approach in Section 4 was to show an individual utilization of at least $\Omega(\tau)$ *on each single processor* after an "overflowed" partitioning phase. However, for RM-TS, we will not do that directly. Instead, we will show the appropriate bound for *sets of processors*.

We first introduce some additional notation. Let's assume that $K \geq 0$ heavy tasks are pre-assigned in the first phase of RM-TS. Then $\mathcal{P}$ is partitioned into the *pre-assigned* processors $\mathcal{P}^{\mathcal{P}} := \{P_1, \ldots, P_K\}$ and the *normal* processors $\mathcal{P}^{\mathcal{N}} := \{P_{K+1}, \ldots, P_M\}$. We also use $\mathcal{P}_{\geq q} := \{P_q, \ldots, P_M\}$ to denote the set of processors with index of at least q.

We want to show that, after a failed partitioning procedure of τ, the total utilization sum of all processors is at least $M \cdot \Omega(\tau)$. We do that inductively:

- **Base Case** (Lemma 10): We show that the total utilization of all normal processors is at least $|\mathcal{P}^{\mathcal{N}}| \cdot \Omega(\tau)$:
$$\sum_{P_q \in \mathcal{P}^{\mathcal{N}}} \mathcal{U}(P_q) \geq |\mathcal{P}^{\mathcal{N}}| \cdot \Omega(\tau)$$
- **Inductive Step** (Lemma 12): For a pre-assigned processor P_m, we infer the bound for processors $\{P_m, ..., P_M\}$ from the bound for processors $\{P_{m+1}, ..., P_M\}$:
$$\sum_{P_q \in \mathcal{P}_{\geq m+1}} \mathcal{U}(P_q) \geq |\mathcal{P}_{\geq m+1}| \cdot \Omega(\tau)$$
$$\Rightarrow \sum_{P_q \in \mathcal{P}_{\geq m}} \mathcal{U}(P_q) \geq |\mathcal{P}_{\geq m}| \cdot \Omega(\tau)$$

The induction implies the expected bound $M \cdot \Omega(\tau)$ with $m = 1$.

Before going into the main proof, we introduce a new concept $\mathcal{P}^{\mathcal{R}}(\tau_i^t)$ and its property, which will be useful later, in both the base case and the inductive steps.

Definition 5. *Given a heavy split task τ_i whose tail subtask τ_i^t is assigned to P_q, we define the* Related Processor Set *of τ_i^t by*

$$\mathcal{P}^{\mathcal{R}}(\tau_i^t) := \begin{cases} \mathcal{P}^{\mathcal{N}} & P_q \text{ is a normal processor} \\ \mathcal{P}_{\geq q} & \text{otherwise} \end{cases}$$

Intuitively, $\mathcal{P}^{\mathcal{R}}(\tau_i^t)$ is the set of processors that might host normal tasks with lower priorities than the heavy task τ_i. We have the following property regarding to the total assigned utilization of $\mathcal{P}^{\mathcal{R}}(\tau_i^t)$.

Lemma 8. *Suppose a* heavy *task τ_i's tail τ_i^t is assigned to P_t. If*

$$\mathcal{U}(\mathcal{P}^{\mathcal{R}}(\tau_i^t)) < |\mathcal{P}^{\mathcal{R}}(\tau_i^t)| \cdot \Omega(\tau) \tag{9}$$

then we have

$$Y_t + U_i^t < \Omega(\tau) - U_i^{body}$$

Proof. Since τ_i is a heavy task but not pre-assigned, it failed the Pre-assign Condition, satisfying the negation of that condition:

$$\sum_{j>i} U_j > (|\mathcal{P}^{\triangleright}(\tau_i)| - 1) \cdot \Omega(\tau) \tag{10}$$

We split the utilization sum of all lower-priority tasks in two parts:

$$\Psi^{\alpha}(\tau_i) := \sum_{\substack{j>i \\ \tau_j \in \tau(\mathcal{P}^{\mathcal{R}}(\tau_i^t))}} U_j$$

$$\Psi^{\beta}(\tau_i) := \sum_{\substack{j>i \\ \tau_j \in \tau(\mathcal{P} \setminus \mathcal{P}^{\mathcal{R}}(\tau_i^t))}} U_j$$

The tasks contributing to $\Psi^{\beta}(\tau_i)$ are all pre-assigned tasks on processors in $\mathcal{P} \setminus \mathcal{P}^{\mathcal{R}}(\tau_i)$, since in the second and third phase of RM-TS, tasks are assigned in increasing priority order and no task will be assigned to $\mathcal{P} \setminus \mathcal{P}^{\mathcal{R}}(\tau_i)$ before all processors in $\mathcal{P}^{\mathcal{R}}(\tau_i)$ are full. We also know that these pre-assigned tasks are all on processors in $\mathcal{P}^{\triangleright}(\tau_i)$, since in the first phase of RM-TS tasks are assigned in decreasing priority order (all processors in $\mathcal{P} \setminus \mathcal{P}^{\triangleright}(\tau_i)$ have been "occupied" by pre-assigned tasks with higher priorities than τ_i and will not host other lower-priority pre-assign tasks). Therefore, we know that all tasks contributing to $\Psi^{\beta}(\tau_i)$ are pre-assigned tasks on processors in $\mathcal{P}^{\triangleright}(\tau_i) \setminus \mathcal{P}^{\mathcal{R}}(\tau_i)$. We further know that each pre-assigned processor has one pre-assigned task, and each task has a utilization of at most $\Omega(\tau)$ (our assumption stated in the beginning of Section 5). Thus, we have:

$$\Psi^{\beta}(\tau_i) \leq (|\mathcal{P}^{\triangleright}(\tau_i)| - |\mathcal{P}^{\mathcal{R}}(\tau_i^t)|) \cdot \Omega(\tau) \tag{11}$$

By replacing $\sum_{j>i} U_j$ by $\Psi^{\alpha}(\tau_i) + \Psi^{\beta}(\tau_i)$ in (10) and applying (11), we get:

$$\Psi^{\alpha}(\tau_i) > (|\mathcal{P}^{\mathcal{R}}(\tau_i^t)| - 1) \cdot \Omega(\tau) \tag{12}$$

So far we have derived a lower bound for $\Psi^{\alpha}(\tau_i)$. In the following, we will also derive an upper bound for it. We consider the utilizations of all processors in $\mathcal{P}^{\mathcal{R}}(\tau_i^t)$. We use the observation that all normal tasks with lower priority than τ_i have already been assigned to some processor in $\mathcal{P}^{\mathcal{R}}(\tau_i^t)$ when assigning τ_i. Recall that as defined in the previous section, we use X_{b_j} to denote the utilization sum of tasks on P_{b_j} with lower priority than τ_i, and X_t the utilization sum of tasks on P_t with lower priority than τ_i. We further use $\mathcal{P}^{\mathcal{F}}(\tau_i)$ to denote the set of processors in $\mathcal{P}^{\mathcal{R}}(\tau_i^t)$ that do *not* hold any part of τ_i. For each processor $P_q \in \mathcal{P}^{\mathcal{F}}(\tau_i)$ we use X_q to denote the utilization sum of tasks on P_q with lower priority than τ_i. With this notation, $\Psi^{\alpha}(\tau_i)$ can be written as:

$$\Psi^{\alpha}(\tau_i) = X_t + \sum_{j \in [1, B_i]} X_{b_j} + \sum_{P_q \in \mathcal{P}^{\mathcal{F}}(\tau_i)} X_q \tag{13}$$

Both sums can be expressed or bounded as follows:

$$\sum_{j \in [1, B_i]} X_{b_j} = \sum_{j \in [1, B_i]} \mathcal{U}(P_{b_j}) - U_i^{body}$$

$$\sum_{p_q \in \mathcal{P}^{\mathcal{F}}(\tau_i)} X_q \leq \sum_{P_q \in \mathcal{P}^{\mathcal{F}}(\tau_i)} \mathcal{U}(P_q)$$

Using both in (13) gives us an upper bound for $\Psi^{\alpha}(\tau_i)$:

$$X_t + \sum_{j \in [1, B_i]} \mathcal{U}(P_{b_j}) - U_i^{body} + \sum_{P_q \in \mathcal{P}^{\mathcal{F}}(\tau_i)} \mathcal{U}(P_q) \geq \Psi^{\alpha}(\tau_i) \tag{14}$$

We combine the lower bound (12) and upper bound (14) to get:

$$X_t + \sum_{j \in [1, B_i]} \mathcal{U}(P_{b_j}) - U_i^{body} + \sum_{P_q \in \mathcal{P}^{\mathcal{F}}(\tau_i)} \mathcal{U}(P_q) > (|\mathcal{P}^{\mathcal{R}}(\tau_i^t)| - 1) \cdot \Omega(\tau) \tag{15}$$

For the last part of the proof, we note that also our general assumption (9) for the lemma can be written using these utilization sums:

$$\mathcal{U}(P_t) + \sum_{j \in [1, B_i]} \mathcal{U}(P_{b_j}) + \sum_{P_q \in \mathcal{P}^{\mathcal{F}}(\tau_i)} \mathcal{U}(P_q) < |\mathcal{P}^{\mathcal{R}}(\tau_i^t)| \cdot \Omega(\tau)$$

By applying this to (15) we have

$$\mathcal{U}(P_t) - X_t < \Omega(\tau) - U_i^{body}$$
$$\Leftrightarrow Y_t + U_i^t < \Omega(\tau) - U_i^{body} \quad (\text{since } \mathcal{U}(P_t) = X_t + U_i^t + Y_t)$$

Proved. □

5.2.1 Base Case

Now we start the main proof with its base case. The proof strategy is: We assume that the total assigned utilization of normal processors is below the expected bound, by which we can derive the absence of bottlenecks on some processors in $\mathcal{P}^{\mathcal{N}}$. This contradicts the fact that there is at least one bottleneck on each processor after a failed partitioning procedure.

First, Lemma 4 still holds for normal processors under RM-TS, i.e., a bottleneck on a normal processor with assigned utilization lower than $\Omega(\tau)$ is neither a non-split task nor a body subtask. This is because the partitioning procedure of RM-TS on normal processors is exactly the same as RM-TS/light and one can reuse the reasoning for Lemma 4 here. In the following, we focus on the difficult case of tail subtasks.

Lemma 9. *Suppose there are remaining tasks after the second phase of* RM-TS. *Let τ_i^t be a tail subtask assigned to P_t. If both the following conditions are satisfied*

$$\sum_{P_q \in \mathcal{P}^{\mathcal{N}}} \mathcal{U}(P_q) < |\mathcal{P}^{\mathcal{N}}| \cdot \Omega(\tau) \tag{16}$$

$$\mathcal{U}(P_t) < \Omega(\tau) \tag{17}$$

then τ_i^t is not a bottleneck on P_t.

Note the difference between Lemma 9 and its counterpart Lemma 6 in RM-TS/light: In Lemma 9, we need both Condition (16) and (17), while in Lemma 6 we only need (16). This is because in RM-TS/light all tasks are assumed to be light, while here we also need to deal with heavy tasks. The proof idea for heavy tasks is to utilize the pre-assign condition, which provides the information on the utilization sum of all the processors in $\mathcal{P}^{\mathcal{N}}$.

Proof. We prove by contradiction: We assume the lemma does *not* hold for one or more tasks, and let τ_i be the lowest-priority one among these tasks.

Similar with the proof of its counterpart in RM-TS/light (Lemma 6), we will first show that all processors hosting τ_i's body subtasks have assigned utilization at least $\Omega(\tau)$. We do this by contradiction. Assume $\mathcal{U}(P_{b_j}) < \Omega(\tau)$. Since τ_i^t does not satisfy the lemma, both of the preconditions of the lemma must be true for τ_i^t, particularly (16) must be true. (16) together with our assumption $\mathcal{U}(P_{b_j}) < \Omega(\tau)$ implies the tail subtasks on P_{b_j} are not bottlenecks (the tail subtasks on P_{b_j} all satisfy this lemma, since they all have lower priorities than τ_i, and by assumption τ_i is the lowest-priority task does not satisfy this lemma). By Lemma 4 (which still holds for normal processors as discussed above), we know a bottleneck of P_{b_j} is neither a non-split task nor a body subtask. So we can conclude that there is no bottleneck on P_{b_j}, which is a contradiction. Therefore, we have proved that all processors hosting τ_i's body subtasks have assigned utilization at least $\Omega(\tau)$.

In the following we will prove τ_i^t is not a bottleneck, by deriving Condition (3) and apply Lemma 5 to τ_i^t. τ_i is either light or heavy. For the case τ_i is light, the proof is exactly the same as for Lemma 6, since the second phase of RM-TS works in exactly the same way as RM-TS/light. Note that to prove for the light task case, only Condition (16) is needed (the same as in Lemma 6).

In the following we consider the case that τ_i is heavy, which is the new challenge of this proof. We distinguish two cases:

- $U_i^{body} \geq \frac{\Omega(\tau)-\Theta(\tau)}{1-\Theta(\tau)}$
  By definition, $\mathcal{P}^{\mathcal{R}}(\tau_i) = \mathcal{P}^{\mathcal{N}}$ since τ_i^t is assigned to a normal processor. And since τ_i is heavy, by Condition (16) and Lemma 8 we have:
  $$Y_t + U_i^t < \Omega(\tau) - U_i^{body} \tag{18}$$
  In order to derive Condition (3) of Lemma 5, which indicates τ_i^t is not a bottleneck, we only need to prove
  $$\begin{aligned}\Omega(\tau) - U_i^{body} &\leq \Theta(\tau)(1-U_i^{body}) \\ \Leftrightarrow (1-\Theta(\tau))U_i^{body} &\geq \Omega(\tau)-\Theta(\tau) \\ \Leftrightarrow U_i^{body} &\geq \frac{\Omega(\tau)-\Theta(\tau)}{1-\Theta(\tau)} \text{ (since } \Theta(\tau)<1)\end{aligned}$$
  which is obviously true due to the precondition of this case. So Condition (3) holds for this case.
- $U_i^{body} < \frac{\Omega(\tau)-\Theta(\tau)}{1-\Theta(\tau)}$
  First, Condition (17) can be rewritten as
  $$X_t + Y_t + U_i^t < \Omega(\tau) \tag{19}$$
  Since all processors hosting τ_i's body subtasks have assigned utilization at least $\Omega(\tau)$ (proved in above), we have
  $$\sum_{j\in[1,B_i]} X_{b_j} + U_i^{body} > B_i \cdot \Omega(\tau)$$
  Since at each step of the second phase of RM-TS, we always select the processor with the minimal assigned utilization to assign the current (sub)task, we have $X_t \geq X_{b_j}$ for each X_{b_j}. Therefore we have
  $$\begin{aligned}B_iX_t + U_i^{body} &\geq B_i\cdot\Omega(\tau) \\ \Rightarrow X_t &\geq \Omega(\tau) - U_i^{body} \text{ (since } B_i \geq 1)\end{aligned}$$
  combining which and (19) we get
  $$Y_t + U_i^t < U_i^{body}$$
  Now, to prove Condition (3), we only need to show
  $$\begin{aligned}U_i^{body} &\leq \Theta(\tau)(1-U_i^{body}) \\ \Leftrightarrow U_i^{body} &\leq \frac{\Theta(\tau)}{1+\Theta(\tau)}\end{aligned}$$
  Due to the precondition of this case $U_i^{body} < \frac{\Omega(\tau)-\Theta(\tau)}{1-\Theta(\tau)}$, we only need to prove
  $$\begin{aligned}\frac{\Omega(\tau)-\Theta(\tau)}{1-\Theta(\tau)} &\leq \frac{\Theta(\tau)}{1+\Theta(\tau)} \\ \Leftrightarrow \Omega(\tau)-\Theta(\tau)+\Theta(\tau)\cdot\Omega(\tau)-\Theta(\tau)^2 &\leq \Theta(\tau)-\Theta(\tau)^2 \\ \Leftrightarrow \Omega(\tau) &\leq \frac{2\Theta(\tau)}{1+\Theta(\tau)}\end{aligned}$$
  which is true since $\Omega(\tau)$ is assumed to be at most $\frac{2\Theta(\tau)}{1+\Theta(\tau)}$ in RM-TS. So in this case Condition (3) also holds.

In summary, in both cases we have proved that Condition (3) holds, and by applying Lemma 5 we know τ_i^t is not a bottleneck. □

Now we are ready to prove the base case:

Lemma 10. *Suppose there are remaining tasks after the second phase of* RM-TS *(there exists at least one bottleneck on each normal processor). We have:*
$$\sum_{P_q\in\mathcal{P}^{\mathcal{N}}} \mathcal{U}(P_q) \geq |\mathcal{P}^{\mathcal{N}}|\cdot\Omega(\tau)$$

Proof. We prove by contradiction. We assume that
$$\sum_{P_q\in\mathcal{P}^{\mathcal{N}}} \mathcal{U}(P_q) < |\mathcal{P}^{\mathcal{N}}|\cdot\Omega(\tau)$$
then there is at least one processor P_q in $\mathcal{P}^{\mathcal{N}}$ such that $\mathcal{U}(P_q) < \Omega(\tau)$. By Lemma 5 (which still holds for processors in $\mathcal{P}^{\mathcal{N}}$ as discussed above) we know a bottleneck of P_q is neither a non-split task nor a body subtask, and by Lemma 9 we know a bottleneck of P_q is not a tail-subtask either. So we can conclude that there is no bottleneck on P_q, which forms the contradiction. □

5.2.2 Inductive Step

We start with a useful property concerning the pre-assigned tasks' local priorities.

Lemma 11. *Suppose P_m is a pre-assigned processor. If*
$$\sum_{P_q\in\mathcal{P}_{\geq m+1}} \mathcal{U}(P_q) \geq |\mathcal{P}_{\geq m+1}|\cdot\Omega(\tau) \tag{20}$$
then the pre-assigned task on P_m has the lowest *priority among all tasks assigned to P_m.*

Proof. Let τ_i be the pre-assigned task on P_m. Since τ_i is pre-assigned, we know that it satisfies the Pre-assign Condition:
$$\sum_{j>i} U_j \leq \underbrace{(|\mathcal{P}^{\triangleright}(\tau_i)|-1)}_{|\mathcal{P}_{\geq m+1}|}\cdot\Omega(\tau)$$
Using this with assumption (20) we have:
$$\sum_{P_q\in\mathcal{P}_{\geq m+1}} \mathcal{U}(P_q) \geq \sum_{j>i} U_j \tag{21}$$
which means the total capacity of the processors with larger indices is enough to accommodate all lower-priority tasks.

By the partitioning algorithm, we know that no tasks, except τ_i which has been pre-assigned already, will be assigned to P_m before all processors with larger indices are full. So we know that no task with priority lower than τ_i will be assigned to P_m. □

Now we start the main proof of the inductive step.

Lemma 12. *We use* RM-TS *to partition task set τ. Suppose there are remaining tasks after processor P_m is full (there exists at least one bottleneck on P_m). If*
$$\sum_{P_q\in\mathcal{P}_{\geq m+1}} \mathcal{U}(P_q) \geq |\mathcal{P}_{\geq m+1}|\cdot\Omega(\tau) \tag{22}$$
then we have
$$\sum_{P_q\in\mathcal{P}_{\geq m}} \mathcal{U}(P_q) \geq |\mathcal{P}_{\geq m}|\cdot\Omega(\tau)$$

Proof. We prove by contradiction. Assume
$$\sum_{P_q\in\mathcal{P}_{\geq m}} \mathcal{U}(P_q) < |\mathcal{P}_{\geq m}|\cdot\Omega(\tau) \tag{23}$$
With assumption (22) this implies the bound on P_m's utilization:
$$\mathcal{U}(P_m) < \Omega(\tau) \tag{24}$$
As before, with (24) we want to prove that a bottleneck on P_m is neither a non-split task, a body subtask nor a tail subtask, which forms a contradiction and completes the proof. In the following we consider each type individually.

We first consider non-split tasks. Again, the deflatable parametric utilization bound $\Omega(\tau)$ is sufficient to guarantee the schedulability of non-split tasks, although the relative deadlines of split subtasks on this processor may change. Thus, (24) implies that a non-split task cannot be a bottleneck of P_m.

Then we consider body subtasks. By Lemma 11 we know the pre-assigned task has the lowest priority on P_m. We also know that all normal tasks on P_m have lower priority than the body subtask, since in the third phase of RM-TS tasks are assigned in increasing priority order. Therefore, we can conclude that the body subtask has the highest priority on P_m, and cannot be a bottleneck.

At last we consider tail subtasks. Let τ_i^t be a tail subtask assigned to P_m. We distinguish the following two cases:

- $U_i^{body} < \frac{\Theta(\tau)}{1+\Theta(\tau)}$
  The inductive hypothesis (22) guarantees with Lemma 11 that the pre-assigned task has the lowest priority on P_m, so X_t contains at least the utilization of this pre-assigned task, which is heavy. So we have:
  $$X_t \geq \frac{\Theta(\tau)}{1+\Theta(\tau)} \tag{25}$$

We can rewrite (24) as $X_t + Y_t + U_i^t < \Omega(\tau)$ and apply it to (25) to get:

$$Y_t + U_i^t < \Omega(\tau) - \frac{\Theta(\tau)}{1+\Theta(\tau)} \qquad (26)$$

Recall that $\Omega(\tau)$ is restricted by an upper bound in RM-TS:

$$\Omega(\tau) \le \frac{2\Theta(\tau)}{1+\Theta(\tau)}$$

$$\Leftrightarrow \Omega(\tau) - \frac{\Theta(\tau)}{1+\Theta(\tau)} \le \Theta(\tau)(1 - \frac{\Theta(\tau)}{1+\Theta(\tau)})$$

$$\Rightarrow \Omega(\tau) - \frac{\Theta(\tau)}{1+\Theta(\tau)} < \Theta(\tau)(1 - U_i^{body}) \left(\text{since } U_i^{body} < \frac{\Theta(\tau)}{1+\Theta(\tau)}\right)$$

And by (26) we have $Y_t + U_i^t < \Theta(\tau)(1 - U_i^{body})$. By Lemma 5 we know τ_i^t is not a bottleneck.

- $U_i^{body} \ge \frac{\Theta(\tau)}{1+\Theta(\tau)}$

  Since $U_i > U_i^{body} \ge \frac{\Theta(\tau)}{1+\Theta(\tau)}$, we know in this case τ_i is a heavy task. By the inductive hypothesis (22) and Lemma 8 we know ($\mathcal{P}^{\mathcal{R}}(\tau_i) = \sum_{P_q \in \mathcal{P}_{\ge m+1}} \mathcal{U}(P_q)$ since P_m is a pre-assigned processor):

$$Y_t + U_i^t < \Omega(\tau) - U_i^{body}$$

$$\Rightarrow Y_t + U_i^t < \frac{2\Theta(\tau)}{1+\Theta(\tau)} - U_i^{body} \quad \left(\text{since } \Omega(\tau) \le \frac{2\Theta(\tau)}{1+\Theta(\tau)}\right)$$

  By the precondition of this case $U_i^{body} \ge \frac{\Theta(\tau)}{1+\Theta(\tau)}$, we have

$$\frac{2\Theta(\tau)}{1+\Theta(\tau)} - U_i^{body} \le \Theta(\tau)(1 - U_i^{body})$$

  Applying this to above we get $Y_t + U_i^t < \Theta(\tau)(1 - U_i^{body})$. By Lemma 5 we know τ_i^t is not a bottleneck.

In summary, we have shown that in both cases the tail subtask τ_i^t is not a bottleneck of P_m. So we can conclude that there is no bottleneck on P_m, which results in a contradiction and establishes the proof. □

5.2.3 Utilization Bound

Lemma 10 (base case) and Lemma 12 (inductive step) inductively proved that after a failed partitioning, the total utilization of all processors is *at least* $M \cdot \Omega(\tau)$. And since there are (sub)tasks in not assigned to any processor after a failed partitioning, τ's normalized utilization $\mathcal{U}_M(\tau)$ is *strictly larger* than $\Omega(\tau)$. So we have:

Lemma 13. *Any task set τ with normalized utilization $\mathcal{U}_M(\tau) \le \Omega(\tau)$ can be successfully partitioned by* RM-TS.

Now we will show that a task set is guaranteed to be schedulable if it is successfully partitioned by RM-TS.

Lemma 14. *If a task set is successfully partitioned by* RM-TS*, the tasks on each processor are schedulable by* RMS.

Proof. Similar with RM-TS/light, the tasks on each *normal* processor are schedulable by RMS, which is guaranteed by the algorithm Assign(τ_i^k, P_q).

On *pre-assigned* processors, Assign(τ_i^k, P_q) guarantees all tasks are can meet deadlines, under the assumption that the body subtask (if there is one) on a pre-assigned processor has the highest priority on that processor. In the following we will show that this assumption is true, and thereby establish the proof.

Let P_q be a pre-assigned processor involved in the third phase of RM-TS, and a body subtask $\tau_i^{b_j}$ is assigned to P_q. By Lemma 10 and 12 we can inductively prove that the total assigned utilization of processors in $\mathcal{P}_{\ge q+1}$ is at least $|\mathcal{P}_{\ge q+1}| \cdot \Omega(\tau)$. And by Lemma 11 we know a pre-assigned task on processors P_q has the lowest priority on that processor, particularly, has lower priority than $\tau_i^{b_j}$. We also know that all other tasks on P_q have lower priority than $\tau_i^{b_j}$, since tasks are assigned in increasing priority order and $\tau_i^{b_j}$ is the last one assigned to P_q. In summary we know a body subtask has the highest priority on its host processor. □

By now we have proved that any task with total utilization no larger than $\Omega(\tau)$ can be successfully partitioned by RM-TS, and all tasks can meet deadline if they are scheduled on each processor by RMS. So we can conclude the utilization bound of RM-TS:

Theorem 15. *Given a deflatable parametric utilization bound $\Omega(\tau) \le \frac{2\Theta(\tau)}{1+\Theta(\tau)}$ derived from the task set τ's parameters. If*

$$\mathcal{U}_M(\tau) \le \Omega(\tau)$$

then τ is schedulable by RM-TS.

6 Conclusions and Future Work

We have developed new fixed-priority multiprocessor scheduling algorithms overstepping the Liu and Layland utilization bound. The first algorithm RM-TS/light can achieve any deflatable parametric utilization bound for light task sets. The second algorithm RM-TS gets rid of the light restriction and work for any task set, if the bound is under a threshold $\frac{2\Theta(\tau)}{1+\Theta(\tau)}$. Further, the new algorithms use exact analysis RTA, instead of the worst-case utilization threshold as in [13], to determine the maximal workload assigned to each processor. Therefore, the average-case performance is significantly improved. As future work, we will extend our algorithms to deal with task graphs specifying dependency constraints and task communication.

References

[1] J. Anderson, V. Bud, and U.C. Devi. An EDF-based scheduling algorithm for multiprocessor soft real-time systems. In *ECRTS*, 2005.

[2] B. Andersson. Global static priority preemptive multiprocessor scheduling with utilization bound 38%. In *OPODIS*, 2008.

[3] B. Andersson, S. Baruah, and J. Jonsson. Static priority scheduling on multiprocessors. In *RTSS*, 2001.

[4] B. Andersson and K. Bletsas. Sporadic multiprocessor scheduling with few preemptions. In *ECRTS*, 2008.

[5] B. Andersson, K. Bletsas, and S. Baruah. Scheduling arbitrary-deadline sporadic task systems multiprocessors. In *RTSS*, 2008.

[6] B. Andersson and E. Tovar. Multiprocessor scheduling with few preemptions. In *RTCSA*, 2006.

[7] T. P. Baker. An analysis of EDF schedulability on a multiprocessor. *IEEE Transaction on Parallel and Distributed Systems*, 2005.

[8] T. P. Baker and S. Baruah. Sustainable multiprocessor scheduling of sporadic task systems. In *ECRTS*, 2009.

[9] S. Baruah and A. Burns. Sustainable schedulability analysis. In *RTSS*, 2006.

[10] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah. *A Categorization of Real-Time Multiprocessor Scheduling Problems and Algorithms*. 2004.

[11] D. Chen, A. K. Mok, and T. W. Kuo. Utilization bound revisited. In *IEEE Transaction on Computers*, 2003.

[12] S. K. Dhall and C. L. Liu. On a real-time scheduling problem. In *Operations Research, Vol. 26, No. 1, Scheduling*, 1978.

[13] N. Guan, M. Stigge, W. Yi, and G. Yu. Fixed-priority multiprocessor scheduling with Liu & Layland's utilization bound. In *RTAS*, 2010.

[14] S. Kato and N. Yamasaki. Semi-partitioned fixed-priority scheduling on multiprocessors. In *RTAS*, 2009.

[15] S. Kato, N. Yamasaki, and Y. Ishikawa. Semi-partitioned scheduling of sporadic task systems on multiprocessors. In *ECRTS*, 2009.

[16] T. W. Kuo and A. K. Mok. Load adjustment in adaptive real-time systems. In *RTSS*, 1991.

[17] K. Lakshmanan, R. Rajkumar, and J. Lehoczky. Partitioned fixed-priority preemptive scheduling for multi-core processors. In *ECRTS*, 2009.

[18] S. Lauzac, R. Melhem, and D. Mosse. An efficient rms admission control and its application to multiprocessor scheduling. In *IPPS*, 1998.

[19] J. P. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *RTSS*, 1989.

[20] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. In *Journal of the ACM*, 1973.

[21] J. W. S. Liu. *Real-time systems*. Prentice Hall, 2000.