Technical Report MPI-SWS-2016-009

Quantifying the Effect of Period Ratios on Schedulability of Rate Monotonic

Mitra Nasri[(1)], Morteza Mohaqeqi[(2)], Gerhard Fohler[(3)]
[(1)] Max Planck Institute for Software Systems (MPI-SWS), Germany.
[(2)] Department of Information Technology, Uppsala University, Sweden.
[(3)] Chair of Real-time Systems, Technische Universität Kaiserslautern, Germany.
mitra@mpi-sws.org, morteza.mohaqeqi@it.uu.se, fohler@eit.uni-kl.de

ABSTRACT

In this paper, we study the effect of period ratio and utilization of the tasks on the schedulability of rate monotonic (RM) in uni-processor systems with preemptive periodic or sporadic tasks. By quantifying this effect, we show that there exist other task sets (other than harmonic tasks in which each period is an integer multiple of the smaller periods) which are RM-friendly, i.e., they can be scheduled by RM up to 100% utilization. Furthermore, in order to quantify non-RM-friendly task sets, we derive a necessary schedulability test for RM. Our results can be used as a set of design hints for system designers during the parameter assignment phase where periods are assigned. We also show how our results can be used to reduce the computational cost of the schedulability analysis if particular properties hold between the periods. From theoretical perspective, our work improves the understanding about outputs of different random task set generation methods. We provide examples to show how the hidden effect of period ratio may lead to an inaccurate judgment about RM schedulability[1].

1. INTRODUCTION

Rate monotonic (RM) is a widely used scheduling algorithm which has been implemented in many real-time operating systems. It is known that RM is not able to schedule many feasible task sets which can potentially be scheduled by the earliest deadline first (EDF) scheduling algorithm [16]. So far, schedulability analysis of RM has been studied in a number of researches by introducing utilization-based tests such as [16], the exact test [1], approximation tests [6, 11], or sufficient tests [18]. These works try to distinguish a schedulable task set from a non-schedulable one, yet they are limited in terms of providing a deeper knowledge about the behavior of the rate monotonic itself.

In [12], it has been shown that if periods are harmonic, i.e., each period is an integer multiple of the smaller periods, then RM can schedule task sets up to 100% utilization. Harmonic task sets have been widely used in industries. Also there are several works showing how to construct harmonic periods from a given set of initial period values [12, 14] or from a given range [17]. However, harmonic task sets are not the only type of *RM-friendly* task sets, i.e., task sets that can be scheduled by RM up to 100% utilization. In [15] a stochastic analysis has been performed to show that when the period ratio approaches to infinity, the maximum schedulable utilization reaches to 1. Both of these results have motivated us to explicitly consider the role of period ratios on RM schedulability.

The timing parameters of a task set are usually described by the period (or the minimum inter-arrival time) and the worst-case execution time (WCET) of the tasks. In this paper, we focus on sporadic task sets with implicit deadlines. During our preliminary experiments we have noticed that the ratio between consecutive periods (assuming periods are sorted in an ascending order) has a significant effect on RM schedulability. Fig. 1 shows the schedulability ratio of randomly generated task sets as a function of the maximum and minimum consecutive period ratio (CPR), denoted by K^{max} and K^{min}, respectively. We have generated the periods with the following rule $T_i = aT_{i-1}$ where a is a random value with uniform distribution which is selected from a range. In Fig. 1-(a), $a \in [1, K^{max}]$ and in Fig. 1-(b), $a \in [K^{min}, 4]$. Here T_{i-1} denotes period of the previous task assuming that tasks are indexed by their period and T_1 is the smallest period. Fig. 1 shows a strong relation between period ratio and schedulability. In this experiment, we have used the exact schedulability test [1] for determining schedulable task sets.

In this paper, we quantify the effect of timing parameters of the task set on the schedulability of RM. For this aim, we adopt some of the existing schedulability tests for RM and use them to describe the schedulability as a function of period ratio, period residual, and utilization of the tasks, where period residual is the residual part of dividing one period by another. We show that if the average period ratio is large or the residual values are either small or large, the chance that the task set is schedulable by RM is significantly high. We identify some other RM-friendly task sets based

[1]The conference version of this extended technical report appears in the proceedings of Real-Time Networks and Systems (RTNS 2016), October 2016, ACM. Sect. 7 and Appendix have been added to the content of the main paper.

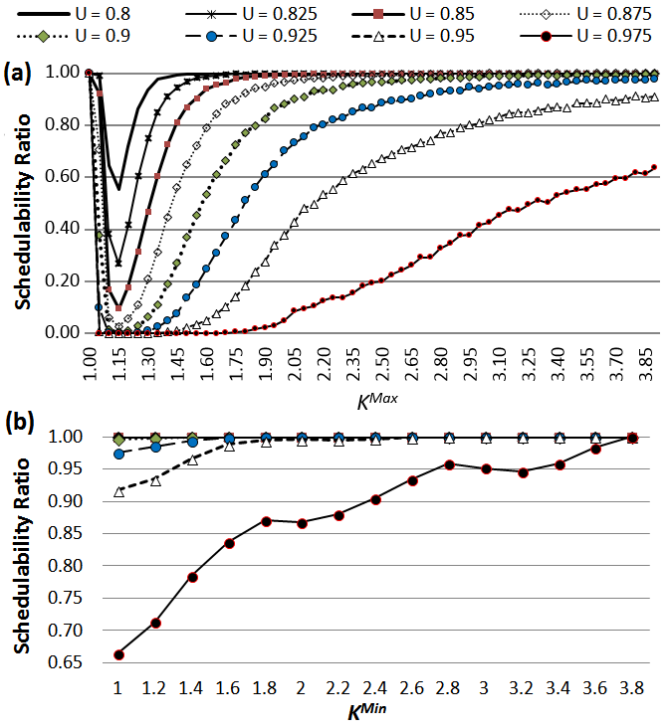

Figure 1: Schedulability ratio as a function of: (a) the maximum CPR, i.e., K^{max}, and (b) the minimum CPR, i.e., K^{min} (see Sect. 6 for details).

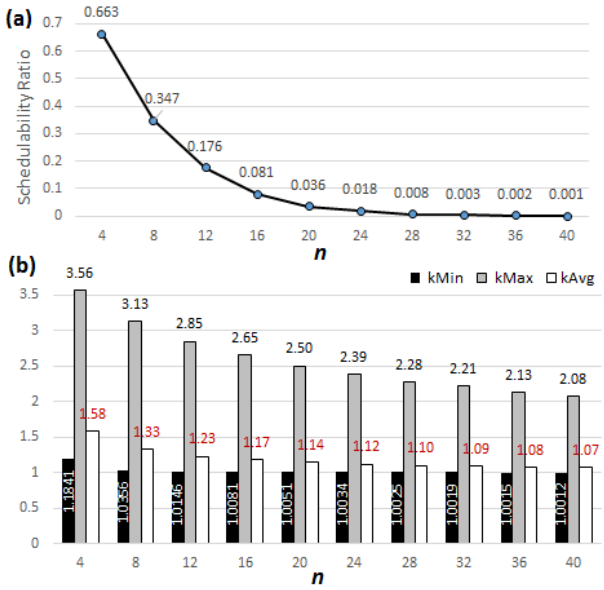

Figure 2: The effect of period selection on RM schedulability. Periods have been selected from range [10, 1000] and utilizations have been generated by uUniFast [4] with total sum 0.9. For each data point, 20,000 task sets have been generated with different number of tasks shown in the horizontal axis. (a) shows schedulability ratio and (b) shows K^{max}, K^{min} and average CPR.

on period ratio. Furthermore, in order to identify non-RM-friendly task sets, we derive a necessary schedulability test for RM. We also show how our results can be used to reduce the computational cost of the exact schedulability analysis if particular properties hold between the periods.

Results of our work can be used by the designer/integrator of the system in order to assign parameters in an RM-friendly way. Doing so, less iterations over the whole design process are needed until the designer reaches to a schedulable set of parameters. Thus, we can reduce both time and cost of the design of large systems with large number of tasks. Moreover, our work can be used to design more efficient task partitioning algorithms for multiprocessor systems.

Knowing how RM schedulability will be affected by the timing parameters of the tasks, we will be able to design more fair experiments when we evaluate performance of the schedulability tests. Also we will have more accurate interpretation of the results. Using the diagram in Fig. 2-(a) we show how lack of this knowledge may lead to an unrealistic conclusion about RM schedulability. In this diagram, we have used the same period assignment method suggested by [4], i.e., the periods have been selected from range [1, 1000] with a uniform distribution. As it can be seen, by the increase in the number of tasks while keeping the utilization 0.9, the schedulability tends to 0. One may interpret the result as "RM is unable to schedule task sets with large number of tasks in 90% utilization". However, by looking at Fig. 2-(b) where K^{max}, K^{min}, and the average CPR of the generated task sets have been reported, we observe that one of the reasons behind such a decrease in schedulability is the decrease in the average CPR. According to Fig. 3-(a), if we change the period assignment such that, for example, the average CPR remains between 1.3 and 1.7, then the resulting schedulability ratio which we get will be different.

The remainder of the paper is organized as follows; System model in described in Sect. 2 and is followed by the motivations of our work in Sect. 3. We study the effect of period ratio on RM schedulability in Sect. 4 and present the necessary schedulability test in Sect. 5. Sect. 6 presents the experimental results. In Sect. 8 related work are presented and finally the paper is concluded in Sect. 9.

2. SYSTEM MODEL AND NOTATIONS

We consider a preemptive uni-processor system with a set of n independent periodic or sporadic tasks. The task set is denoted by $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$. Each task τ_i, $1 \leq i \leq n$, is identified by $\tau_i : (C_i, T_i)$, where $C_i \in \mathbb{R}$ is the worst-case execution time (WCET) and $T_i \in \mathbb{R}$ is the minimum inter-arrival time (called period). We consider implicit deadlines, i.e., deadline of the task is equal to its period. The system utilization is denoted by $U = \sum_{i=1}^{n} u_i$ where $u_i = C_i/T_i$ is the utilization of task τ_i. The *hyperperiod* is the least common multiple (LCM) of the periods. The tasks are indexed according to their period so that $T_1 \leq T_2 \leq \ldots \leq T_n$.

Our work is focused on RM, a fixed priority scheduling algorithm in which priorities are monotonic with $1/T_i$. A set of sporadic tasks is schedulable by RM if the worst-case response time (WCRT) of each task is not greater than its deadline. The WCRT of τ_i can be obtained when all high priority tasks are released synchronously with the release of τ_i and they are activated periodically.

Next, we define relative period ratio and its components.

DEFINITION 1. *Relative period ratio (RPR) of two tasks τ_i and τ_j $(1 \leq j < i \leq n)$ is denoted by $K_{i,j}$ as*

$$K_{i,j} = k_{i,j} + \gamma_{i,j} = \frac{T_i}{T_j} \tag{1}$$

where $k_{i,j} = \lfloor T_i/T_j \rfloor$ is called the base and $\gamma_{i,j} = T_i/T_j - \lfloor T_i/T_j \rfloor$ is called the residual. Notice that $\gamma_{i,j} \in [0, 1)$.

If in Definition 1 we have $i = j + 1$, namely, τ_i and τ_j are two consecutive tasks in the task set, $K_{i,i-1}$ represents the consecutive period ratio (CPR). The period ratio of the task set is defined as $K^{min} = \min\{K_{i,i-1}\}_{i=2}^{n}$. The task set is harmonic if $\forall i, 1 < i \leq n, K_{i,i-1} \in \mathbb{N}$. If for two tasks τ_i and τ_j $(T_j \leq T_i)$, $\gamma_{i,j} = 0$, their period is harmonic. Based on this property, we define the set of *harmonic tasks* with priority higher than or equal to τ_i as

DEFINITION 2. *The set of harmonic tasks with task τ_i is denoted by h_i and defined as*

$$h_i = \{\tau_j | \gamma_{i,j} = 0, 1 \leq j \leq i\} \tag{2}$$

3. MOTIVATIONS

In the pioneer work of Liu and Layland [16], it has been shown that the hardest-to-schedule implicit deadline task set for RM has the following properties: a) $T_1 < T_2 < \ldots < T_n \leq 2T_1$, and b) $C_i = T_{i+1} - T_i$, $1 \leq i < n$, and $C_n = 2T_1 - T_n$. The first condition means that $K_{i,1} < 2$, $\forall i; 1 < i \leq n$. Ironically, however, the hardest-to-schedule task set is the fastest to be checked by the exact schedulability test. According to [15], if for every task τ_i we can find a value $t \leq T_i$ that satisfies the following inequality, the task set (defined in Sect. 2) is schedulable

$$t \geq C_i + \sum_{1 \leq j < i} \left\lceil \frac{t}{T_j} \right\rceil C_j \tag{3}$$

In order to apply the exact schedulability test for the hardest-to-schedule task set, we need to verify only one time instant per task, i.e., $t = T_1 - \epsilon$ where ϵ is a small positive number. This observation motivates us to investigate the relation between the computational complexity of the exact schedulability test and the period ratio of the tasks.

Values of t which must be checked in (3) are limited to the integer multiples of periods of the higher priority tasks than τ_i because the right-hand-side of the equation varies only when $t = aT_j$ where $a \in \mathbb{N}$ and $aT_j \leq T_i$. Following the work of Lehoczky et al., [15], we obtain an upper-bound on the computational complexity of the exact analysis. The following formula counts the number of jobs of high priority tasks within the period of τ_i for which (3) must be checked

$$\theta_i = \sum_{j=1}^{i-1} k_{i,j} \Rightarrow \theta_i \leq (i-1)k_{i,1} \leq (i-1)k_{n,1} \tag{4}$$

where $k_{i,j}$ represents the maximum number of jobs of any task τ_j within one release of τ_i. Since periods are sorted, we have $k_{i,j} \leq k_{i,1} \leq k_{n,1}$ which means that for each task such as τ_i, and for each high priority task such as τ_j, $(j < i)$, we need to verify at most $k_{n,1}$ time instants. In order to evaluate the schedulability of the task set, we must verify the schedulability of each task τ_i, $1 \leq i \leq n$, which means that we will have $\theta = \sum_{i=1}^{n} \theta_i \leq \sum_{i=1}^{n} (i-1)k_{n,1}$ which implies that

$$\theta \leq k_{n,1} \sum_{i=1}^{n} (i-1) \leq k_{n,1} \times \frac{n(n-1)}{2} \leq k_{n,1} \times n^2 \tag{5}$$

Since for each point we must verify (3) which has a complexity of $O(n)$, the final complexity will be $O(k_{n,1} \times n^3)$.

If the maximum ratio between the largest and smallest periods is limited to a reasonable number, the computational complexity of the exact test will remain polynomial-time. This observation justifies why the exact schedulability test for rate monotonic performs very well in practice. In some works, e.g., some experiments in [9], periods have been selected randomly from a fixed range such as $[100, 500]$ or $[100, 300]$, and hence, $k_{n,1}$ will be at most 5 in the generated task sets. Thus, even if the task set has 50 tasks, the maximum number of values which must be checked in (3) would be less than a million. Today's computers can easily perform this evaluation in a reasonably short amount of time. On the other hand, if $k_{n,1}$ is an exponential value with respect to n, for example, imagine that $k_{n,1} \approx 2^n$ (that may happen if the periods have a wide range from a millisecond to tens of seconds), we may have exponential number of operations to verify the schedulability. However, interestingly, we have observed that the schedulability ratio of a task set with large CPR is meaningfully higher than those with small CPR. As it is shown in Fig. 1-(a), with the increase in CPR, the schedulability ratio increases significantly even in high utilizations.

When CPR increases, the ratio between the largest and the smallest period in the system increases as well. It, in turn, increases the duration of time where the decisions made by RM will be the same as EDF for each task. For example, if $T_1 = 2$ and $T_2 = 49$, then for about 47 units of time, EDF and RM both prioritize jobs of τ_1 over the job of τ_2. It means that the duration of time where RM takes optimal decisions (those that do not threaten the schedulability of τ_2) will be increased to 47 over 49 units of time. Moreover, if $u_1 + u_2$ is not a relatively large number, the chance that WCRT of τ_2 happens after time 48 when the latest job of τ_1 is released, decreases.

From another perspective, quantifying the effect of period ratio on RM schedulability helps us to have a better understanding about the results of different task set generation methods. One example is shown in Fig. 2 where periods are selected from a fixed range (e.g., in papers such as [4, 9]) while the number of tasks selected from that range was increasing. If instead of decreasing average CPR (in Fig. 2-(b)), we could have a relatively fixed average CPR, then we had totally different diagram as shown in Fig. 3-(a) (note that these figures show the result of exact RM schedulability test). This time, we have fixed the average CPR, however, since with the increase in the number of tasks, the average task utilization decreases, utilization of each task reduces as well, and hence, we will have task sets with small u_i values (as shown in Fig. 3-(c)). In this case, C_i is small and hence, most of the tasks can be finished early. As a result, the adverse effect of prioritizing a job with *smaller period* over a job with an *earlier deadline* will not cause deadline misses for low priority tasks.

By quantifying the above mentioned effects on the RM schedulability, we will be able to provide design hints for system designers when they configure the system. Conse-

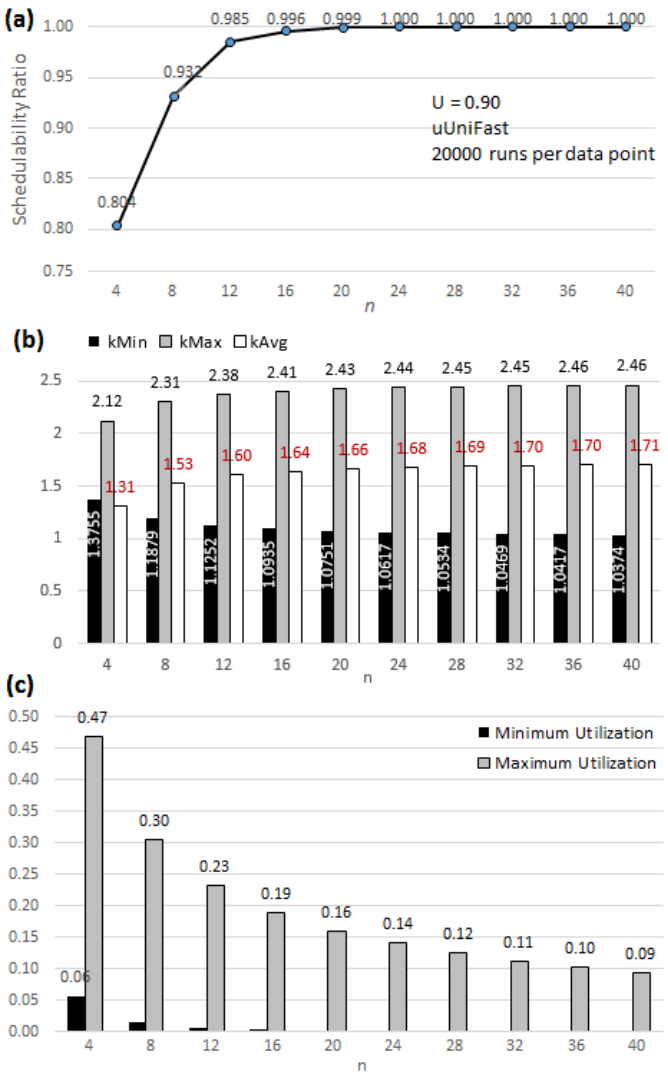

Figure 3: Relative period ratio have been generated as a real number from range [1, 2.5] and utilizations have been generated by uUniFast with sum 0.9**.**

quently, the number of iterations needed to achieve a schedulable configuration will reduce since the chance that the system is already schedulable is higher. Moreover, we are interested to derive conditions which help to reduce the computational cost of schedulability analysis, e.g., by identifying the cases where amortized cost of verifying schedulability of a task will be $O(1)$. This also helps system designers to reduce the time required to find the best configuration.

4. THEEFFECT OF PERIOD RATIO

In this section, we study the effect of period ratio on RM schedulability. First we show the effect of base period ratio, i.e., $k_{i,j}$, and then we discuss the effect of residual, i.e., $\gamma_{i,j}$.

4.1 The Effect of Base Period Ratio

Under the RM scheduling algorithm, the exact WCRT of a task such as τ_i is the minimum $t \geq C_i$ which satisfies (3). τ_i is schedulable if $t \leq T_i$. In some works such as [18] a sufficient schedulability test is used that only verifies the schedulability in a certain set of points in time, i.e., $t \in \{A_{1,i}, A_{2,i}, \ldots, A_{i,i}\}$ where

$$A_{i,j} = \left\lfloor \frac{T_i}{T_j} \right\rfloor T_j \tag{6}$$

In order to quantify the relation between the schedulability and the period ratio, we consider $t = T_i$ and replace it in (3). The following inequality presents a sufficient schedulability condition for τ_i

$$T_i \geq C_i + \sum_{1 \leq j < i} \left\lceil \frac{T_i}{T_j} \right\rceil C_j \tag{7}$$

In the next step, we replace T_i/T_j in (7) by its equivalent value from (1) as $T_i \geq C_i + \sum_{1 \leq j < i} \lceil k_{i,j} + \gamma_{i,j} \rceil C_j$ which leads to

$$T_i \geq \sum_{\substack{1 \leq j < i \\ \tau_j \notin h_i}} (k_{i,j} + 1) C_j + \sum_{\substack{1 \leq j \leq i \\ \tau_j \in h_i}} k_{i,j} C_j \tag{8}$$

where $k_{i,i} = 1$ by definition. Next, we divide (8) by T_i

$$1 \geq \sum_{\substack{1 \leq j < i \\ \tau_j \notin h_i}} \frac{(k_{i,j} + 1)}{T_i} C_j + \sum_{\substack{1 \leq j \leq i \\ \tau_j \in h_i}} \frac{k_{i,j}}{T_i} C_j \tag{9}$$

Using (1) we replace T_i with $(k_{i,j} + \gamma_{i,j})T_j$ in (9) as follows

$$1 \geq \sum_{\substack{1 \leq j < i \\ \tau_j \notin h_i}} \left(\frac{k_{i,j} + 1}{k_{i,j} + \gamma_{i,j}} \right) u_j + \sum_{\substack{1 \leq j \leq i \\ \tau_j \in h_i}} u_j \tag{10}$$

Note that since in the second summation in (9) we have $\tau_j \in h_i$, for every task τ_j, residual is zero, i.e., $\gamma_{i,j} = 0$. Hence, by replacing T_i with $(k_{i,j} + \gamma_{i,j})T_j$, only $k_{i,j}T_j$ remains in the denominator. Next we simplify (10) by replacing $k_{i,j} + 1$ with $k_{i,j} + 1 + \gamma_{i,j} - \gamma_{i,j}$ which implies

$$1 \geq \sum_{\substack{1 \leq j < i \\ \tau_j \notin h_i}} \left(\frac{1 - \gamma_{i,j}}{k_{i,j} + \gamma_{i,j}} \right) u_j + \sum_{1 \leq j \leq i} u_j. \tag{11}$$

The resulting (11) is a function of the tasks' utilization, as well as the base and the residual of the period ratio. As it can be observed in (11), the only tasks with extra negative effect on the schedulability of τ_i are those which are not harmonic with the period of τ_i. This observation is consistent with the results of [14] where it has been shown that to analyze the schedulability of a task set which has m harmonic chains, it is enough to consider each chain as one task (note that a harmonic chain is a set of tasks with harmonic periods.) Consequently, the number of substantial tasks which must be considered in the schedulability analysis reduces to m.

Next we quantify the effect of base period ratio on the schedulability of τ_i. For this aim, we plot $\alpha_{i,j}$ for different values of $k_{i,j}$ from 1 to 5, where

$$\alpha_{i,j} = \frac{1 - \gamma_{i,j}}{k_{i,j} + \gamma_{i,j}} \tag{12}$$

As it can be seen in Fig. 4, for any value of $k_{i,j},$, the maximum value of $\alpha_{i,j}$ appears at $\gamma_{i,j} \approx 0$ because this function is strictly decreasing in the range $\gamma_{i,j} \in [0, 1)$. In fact, $\alpha_{i,j}$ can be interpreted as a multiplier which shows the adverse effect of the utilization of τ_j on the schedulability of τ_i. For example, if $k_{i,j} = 3$, the maximum value of $\alpha_{i,j}$ will

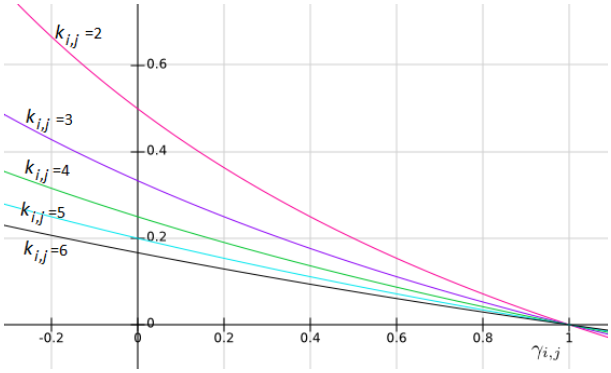

Figure 4: $\alpha_{i,j}$ as a function of $\gamma_{i,j}$ for different values of $k_{i,j}$.

be 0.33, which means that in the worst-case, τ_j adversely adds 33% of its u_j to the total sum of the right-hand-side of (11). However, with the increase in $k_{i,j}$, this effect becomes smaller.

THEOREM 1. *Given task set τ defined in Sect. 2 and K^{min} which is the minimum consecutive period ratio (defined in Sect. 2), a task such as τ_i is schedulable if*

$$1 - \sum_{1 \le j \le i} u_j \ge \sum_{\substack{1 \le j < i \\ \tau_j \notin h_i}} \frac{u_j}{(K^{min})^{i-j}} \tag{13}$$

PROOF. If the minimum period ratio between every two consecutive tasks is at least K^{min}, it means that $k_{i,j} \ge (K^{min})^{i-j}$. As shown in Fig. 4, the $\alpha_{i,j}$ function is strictly decreasing and its maximum value is at $\gamma_{i,j} = 0$, which implies $\alpha_{i,j} = 1/k_{i,j}$, hence, the fraction $(\alpha_{i,j})$ in (11) can be replaced by $1/k_{i,j}$. Thus, based on the fact that $k_{i,j} \ge (K^{min})^{i-j}$, (11) can be simplified to (13) which proves the claim. □

For example, if K^{min} is 4 and we have $\forall j, 1 \le j < i; \tau_j \notin h_i$, then (13) will be

$$\frac{u_{i-1}}{4} + \frac{u_{i-2}}{4^2} + \ldots + \frac{u_1}{4^{i-1}} + \sum_{1 \le j \le i} u_j \le 1 \tag{14}$$

In the next step, we show that with the increase in K^{min}, the chance that the task is schedulable tends to 1.

PROPERTY 1. *In a given task set τ defined in Sect. 2 with $U \le 1$, if $k_{i,j}$ $(\forall j, 1 \le j < i)$ or K^{min} tends to infinity, the chance that the task set is schedulable by RM tends to 1.*

PROOF. If $k_{i,j}$ (or K^{min}) tends to infinity, $\alpha_{i,j}$ tends to 0 because $k_{i,j}$ appears in the denominator of (12). Consecutively, (11) becomes $1 \ge \sum_{1 \le j \le i} u_j$ which is true according to the assumption. It means that both necessary and sufficient schedulability conditions of RM hold, and hence, the task set is schedulable. □

Property 1 shows that having large period ratio increases the schedulability of the task set. This property can be justified by the fact that for a task such as τ_i, the scheduling decisions of RM and EDF are identical until the latest release of any task τ_j, $(1 \le j < i)$ before T_i, i.e., before $A_{j,i}$. If the base period ratio, i.e., $k_{i,j}$ is large, the chance that the task is finished before the last release of the high priority task increases. Thus, RM decisions will not adversely affect the schedulability of τ_i.

4.2 The Effect of Period Residual

In this subsection, we quantify the effect of period residual on the schedulability of RM. First we show the effect of large residual values, i.e., when $\gamma_{i,j}$ tends to 1.

PROPERTY 2. *In a given task set τ defined in Sect. 2 with $U \le 1$, if $\gamma_{i,j}$ tends to 1 $(\forall j, 1 \le j < i)$, the chance that the task set is schedulable by RM tends to 1.*

PROOF. If $\gamma_{i,j}$ tends to 1, $\alpha_{i,j}$ in (12) tends to 0 because regardless of the value of $k_{i,j}$, $1 - \gamma_{i,j}$ tends to 0. Thus, (11) simplifies to $U \le 1$, which already holds due to our assumption. As a result, the task set becomes schedulable by RM. □

Having $\gamma_{i,j} \approx 1$ means that the resulting residual from dividing T_i by any T_j, $1 \le j < i$, must be close to 1. Equivalently, the latest releases of the high priority tasks before T_i must be as far as possible from T_i. In other words, the next release of the high priority tasks after T_i must be very close to T_i, which means that the period of high priority tasks must be almost harmonic with T_i.

Another observation in (12) is that since $k_{i,j}$ appears in the denominator of $\alpha_{i,j}$, if $k_{i,j}$ is large, the contribution of $\gamma_{i,j}$ in the value of $\alpha_{i,j}$ becomes small. Thus, the lowest priority task among those which satisfy $\tau_j \notin h_i$ plays an important role in the schedulability of τ_i because it will have the smallest $k_{i,j}$.

Remark. *To increase RM schedulability, it is better to not have high utilization tasks with close yet non-harmonic periods, because then $k_{i,j}$ will not help in reducing the effect of $u_{i,j}$ in the right-hand-side of (11) which adversely affects the schedulability. Thus, if a task such as τ_i has high utilization, it is better that the next task has large $k_{i+1,i}$.*

In the next step, we show the effect of having small value of $\gamma_{i,j}$ on the schedulability of τ_i. For this aim, first we design another sufficient schedulability test. In this test, we consider the latest release of each high priority task such as τ_j before T_i, denoted by $A_{i,j}$ (defined in (6)). Also we define $A_{i,m} = \min\{A_{i,j}\}_{1 \le j < i}$ and it describes the earliest release among the latest releases of the high priority tasks before T_i. Fig. 5-(a) shows an example with 5 tasks and their latest release before T_i.

Next we rewrite (3) considering $t = A_{i,m}$ as

$$A_{i,m} \ge C_i + \sum_{1 \le j < i} \left\lceil \frac{A_{i,m}}{T_j} \right\rceil C_j \tag{15}$$

Since by definition, $A_{i,m} \le A_{i,j}$ $(\forall j; 1 \le j < i)$, no other high priority task has released its latest job before T_i earlier than $A_{i,m}$. It means that total workload of τ_j before $A_{i,m}$ is smaller than or equal to $\lceil A_{i,m}/T_j \rceil C_j$ which is smaller than $\lfloor T_i/T_j \rfloor C_j$. Hence it is safe to replace $\lceil A_{i,m}/T_j \rceil C_j$ with $\lfloor T_i/T_j \rfloor C_j$ in (15). Next, we replace $A_{i,m}$ in the left-hand-side of (15) by its equivalent value $k_{i,m}T_m$, and then we divide both sides of the inequality by T_i, thus, we have

$$k_{i,m}\frac{T_m}{T_i} \ge u_i + \sum_{1 \le j < i} \frac{u_j T_j k_{i,j}}{T_i} \Rightarrow$$

$$\frac{k_{i,m}}{k_{i,m} + \gamma_{i,m}} \ge u_i + \sum_{1 \le j < i} \frac{k_{i,j} + (\gamma_{i,j} - \gamma_{i,j})}{k_{i,j} + \gamma_{i,j}} u_j \Rightarrow$$

$$\frac{k_{i,m}}{k_{i,m} + \gamma_{i,m}} + \sum_{1 \le j < i} \frac{\gamma_{i,j}}{k_{i,j} + \gamma_{i,j}} u_j \ge \sum_{1 \le j \le i} u_j \tag{16}$$

PROPERTY 3. *In a given task set τ defined in Sect. 2 with $U \leq 1$, if $\gamma_{i,j}$ tends to 0 ($\forall i, j, 1 \leq j < i$), the chance that the task set is schedulable by RM tends to 1.*

PROOF. If $\gamma_{i,j}$ tends to 0, $\gamma_{i,j}/(k_{i,j} + \gamma_{i,j})$ tends to 0. In addition, since τ_m is also one of the high priority tasks, $\gamma_{i,m}$ tends to zero as well. As a result, $k_{i,m}/(k_{i,m} + \gamma_{i,m})$ tends to 1, and hence, (16) becomes $\sum u_j \leq 1$, which is correct due to our assumption. Thus, τ becomes schedulable. □

Property 2 together with Property 3 show that task sets which are *almost harmonic*, i.e., all high priority tasks have a release just before or just after the deadline of τ_i, have a high schedulability ratio, and hence, they are RM-friendly. This situation can easily happen if period of a task such as T_i is an integer multiple of the hyperperiod of the high priority tasks, i.e., $T_i = x \times LCM(T_1, T_2, \ldots, T_{i-1})$ where $x \in \mathbb{N}^{>1}$ is an arbitrary multiplier.

COROLLARY 1. *In a given task set τ defined in Sect. 2 with $U \leq 1$ if for task τ_i we have $\gamma_{i,j} = 0, \forall j; 1 \leq j < i$ and all high priority tasks are schedulable, then τ_i will be schedulable by RM iff*

$$\sum_{j=1}^{i} u_j \leq 1 \tag{17}$$

PROOF. If (17) does not hold, the task is not schedulable because it violates the necessary schedulability condition in [16]. Considering $\gamma_{i,j} = 0, \forall j; 1 \leq j < i$, (16) will be equivalent to (17) which proves the *if* part of the claim. □

Corollary 1 shows that not only tasks with harmonic periods, but also tasks with periods which are harmonic with the LCM of the other periods (even if those periods are not harmonic with each other) will be schedulable by RM if $U \leq 1$. This type of task sets can be found in automotive industry. For example, as shown in a benchmark task set from Autosar [13], period of each periodic task is one of the following values $\{1, 2, 5, 10, 20, 50, 100, 200, 1000\}$ (numbers are in millisecond). Even though 2 and 5, or 20 and 50 are not harmonic, all other periods are integer multiples of 10 or 50. Thus using Corollary 1 we can analyze the schedulability of such low priority tasks (through (17)) efficiently because then the *amortized cost* of schedulability analysis of such tasks will be $O(1)$.

The importance of Properties 1, 2, and 3 becomes more evident when we consider the fact that the task sets with large period ratio, or very small (or large) residual, usually have a large hyperperiod which happens due to nonalignment of the integer multiples of the periods. Adversely, for these task sets, the exact schedulability analysis may have considerable computational cost due to the large number of time instants which must be evaluated.

5. NECESSARY SCHEDULABILITY TEST

In this section, we provide a necessary schedulability condition for RM as a function of utilization of the tasks and period ratios. We show that this test is tighter than the existing $U \leq 1$. Moreover, we derive a condition, based on period residual, for which the test becomes exact. Results of this section are only valid for sporadic tasks or periodic tasks with synchronous release because in these cases, the worst-case release scenario is to have a synchronous periodic release pattern.

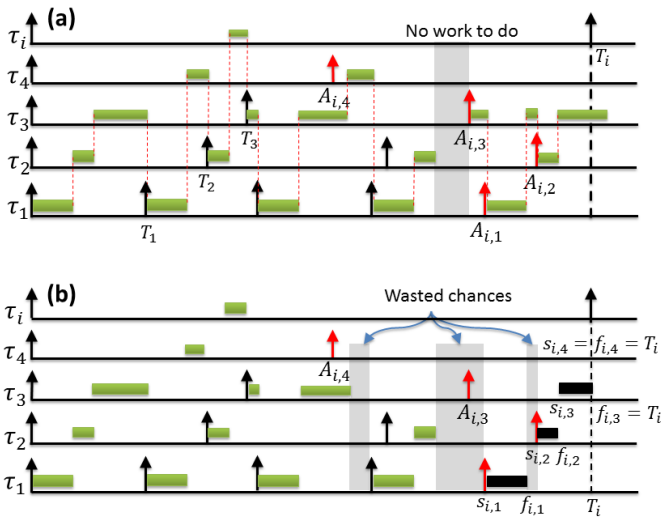

Figure 5: An example to show how the ω_i is calculated. In (a) the true RM schedule is presented while (b) shows the way we calculate ω_i. In (b), τ_3 and τ_4 are planned to start after $f_{i,2}$ even though in reality they could have started earlier. In this example, $f_{i,3} = s_{i,4} = f_{i,4} = T_i$.

We derive a necessary schedulability condition for each task such as τ_i by calculating the lower bound of the *effective workload* of all of the tasks that must be finished before T_i. This lower bound is denoted by Ω_i, and it has 3 components: a) the workload of τ_i which is equal to C_i, b) the exact workload of the high priority jobs which their absolute deadline is smaller than or equal to T_i, and c) the lower bound of the effective workload of the high priority jobs which must be executed until T_i and their absolute deadline is larger than T_i. The last component is denoted by ω_i.

In Fig. 5-(a), the RM schedule for a task set is shown. According to this schedule, the last job of τ_3 cannot finish its execution before T_i, however, τ_i is schedulable since its worst-case response time is equal to T_3. In Fig. 5-(b), we show the 3 components of Ω_i. The execution times which are denoted in black boxes show the lower bound of the contribution of each high priority task in ω_i. As it can be seen, this lower bound is smaller than the actual contribution of the tasks; for example, in Fig. 5-(a), the last job of τ_4 is scheduled slightly after $A_{i,4}$ while in Fig. 5-(b), its contribution to ω_i is 0. In this figure, $s_{i,j}$ and $f_{i,j}$ are the upper bound on the start time and finish time of the last jobs of the high priority tasks.

LEMMA 1. *Given task set τ, the exact workload of the high priority jobs with absolute deadlines smaller than or equal to T_i in the interval $[0, T_i]$ is*

$$\sum_{1 \leq j \leq i} \left\lfloor \frac{T_i}{T_j} \right\rfloor C_j \tag{18}$$

PROOF. Since in a successful schedule there is no deadline miss, all high priority jobs must be executed before their absolute deadlines. Thus, the exact workload from the higher

priority tasks can be obtained by summing up the execution times of every job of those tasks which has its absolute deadline before time T_i. Since each task such as τ_j, $1 \leq j \leq i$ can have at most $\lfloor T_i/T_j \rfloor$ jobs in the interval of time from 0 to T_i, the maximum workload of these jobs is (18) which proves the claim. □

Using (18) we calculate Ω_i as

$$\Omega_i = \sum_{1 \leq j \leq i} \left\lfloor \frac{T_i}{T_j} \right\rfloor C_j + \omega_i \tag{19}$$

LEMMA 2. *Given task set τ, the exact workload of the last job of task τ_1 which is released before T_i and has an absolute deadline larger than T_i is*

$$\min\{C_1, T_i - A_{i,1}\} \tag{20}$$

where $A_{i,1}$ is obtained from (6).

PROOF. Since τ_1 has the highest priority in each of its releases, the actual start time of the last job of τ_1 will be at $A_{i,1}$. Also since it remains the highest priority task, it can be executed until $A_{i,1} + C_1$, however, if $C_1 > T_i - A_{i,1}$, this job will be finished after T_i. Due to the fact that ω_i must only include the workload which must be finished before T_i, the maximum contribution of τ_1 to ω_i cannot be larger than $\min\{C_1, T_i - A_{i,1}\}$, which proves the claim. □

As it can be seen in Fig. 5-(b), the last job of a task such as τ_4 at $A_{i,4}$ can be released much earlier than $A_{i,1}$, and hence, it might have some interference with other jobs of τ_1 or other higher priority tasks. Due to this interference, τ_4 cannot use the whole interval $[A_{i,4}, T_i]$ for its execution since it may not always be the highest priority task in this interval. In other words, calculating the exact amount of time that the last job of a task such as τ_j $(1 < j < i)$ is executed in the interval $[A_{i,j}, T_i]$ requires obtaining the exact interference from the higher priority tasks in that interval. Since this problem is not easy to solve, we simplify the condition to derive the lower bound of the workload instead of the exact amount of the required workload. This lower bound can be obtained by assuming that τ_j does not find any chance to be scheduled before the finish time of τ_{j-1} at $f_{i,j-1}$. This situation has been shown in Fig. 5-(b) where the last job of τ_3 started at $f_{i,2}$. The intervals marked by the *wasted chances* are the price we pay for simplifying our necessary test.

LEMMA 3. *Given task set τ, the lower bound of the effective workload of the last job of each task such as τ_j, $1 < j < i$ that is released before T_i and its absolute deadline is larger than T_i is $f_{i,j} - s_{i,j}$ where $f_{i,j}$ and $s_{i,j}$ are the upper bounds on the finish time and start time of the execution of the last job of τ_j in the interval $[A_{i,j}, T_i]$ and they are calculated as*

$$s_{i,j} = \max\left\{f_{i,j-1}, A_{i,j}\right\} \tag{21}$$

$$f_{i,j} = \min\left\{C_j, T_i - s_{i,j}\right\} + s_{i,j} \tag{22}$$

where $s_{i,1} = A_{i,1}$ and $f_{i,1} = \min\left\{C_1, T_i - s_{i,1}\right\} + s_{i,1}$.

PROOF. The proof is based on induction where we show that the lower bound of the effective workload of any task such as τ_j follows $f_{i,j} - s_{i,j}$.

The base: we show the claims hold for τ_1. Since τ_1 has the highest priority, it executes right after it is released at $A_{i,1}$, and hence, the upper bound of its start time is equal to its actual start time at $s_{i,1} = A_{i,1}$. Also according to Lemma 2, the exact workload of τ_1 before T_i follows (20) which means that the actual finish time of τ_1 is at $f_{i,1} = s_{i,1} + \min\{C_1, T_i - A_{i,1}\}$. Thus, the claims have been proven for τ_1.

The assumption: we assume that $s_{i,m}$ and $f_{i,m}$ obtained from (21) and (22) are the upper bound on the start and finish time of the last job of τ_m, respectively. Also $f_{i,m} - s_{i,m}$ is the lower bound of the effective workload of τ_m which must be scheduled before T_i according to RM scheduling algorithm.

The induction step: we show that $s_{i,m+1}$ and $f_{i,m+1}$ obtained from (21) and (22) are the upper bound on the start and finish time of the last job of τ_{m+1}, respectively. Also $f_{i,m+1} - s_{i,m+1}$ is the lower bound of the effective workload of τ_{m+1} which must be scheduled before T_i.

Before we start the proof, we show some properties about $f_{i,m}$. Because of the way $A_{i,m}$ is defined in (6), $T_i - A_{i,j}$ never becomes negative, thus, $\forall j$, $f_{i,j} \geq 0$. Moreover, directly from (22) we have $\forall j$, $f_{i,j} \leq T_i$. Finally, due to the recursive nature of (22), we have

$$A_{i,1} \leq f_{i,1} \leq f_{i,2} \leq \ldots \leq f_{i,m} \leq T_i \tag{23}$$

because, for example, by replacing $s_{i,m}$ in (22) we have

$$f_{i,m} = \min\left\{C_m, T_i - s_{i,m}\right\} + \max\left\{f_{i,m-1}, A_{i,m}\right\}$$

which means that

$$f_{i,m-1} \leq f_{i,m}$$

From (23) we can conclude that at $f_{i,m}$, all of the higher priority tasks than τ_{m+1} have been executed, otherwise $f_{i,m} = T_i$. If $f_{i,m} = T_i$, then a safe upper bound on $s_{i,m+1}$ will be T_i, which further leads to have a safe upper bound on the finish time of τ_{m+1} at T_i. Since our safe upper bounds fall out of the interval $[A_{i,m+1}, T_i]$, the effective workload of τ_{m+1} will be 0. On the other hand, if $f_{i,m} < T_i$, task τ_{m+1} becomes the highest priority task with a pending job at time $f_{i,m}$ and will have a chance to be scheduled by RM. If τ_{m+1} is already released at that time, i.e., $A_{i,m+1} \leq f_{i,m}$, a safe upper bound on its start time will be $s_{i,m+1} = f_{i,m}$ because then there will be no other high priority task in the system and $T_i - s_{i,m+1} > 0$. In such situation, RM has no other choice but to schedule τ_{m+1}. Otherwise, if $A_{i,m+1} > f_{i,m}$, since none of the tasks with higher priority than τ_{m+1} will release a job after $f_{i,m}$, the exact upper bound on the start time of τ_{m+1} can be obtained at $s_{i,m+1} = A_{i,m+1}$. The latter claim can be proven by considering the fact that due to (22) and (21), $\forall j, 1 \leq j < m; A_{i,j} \leq f_{i,j} \leq f_{i,m}$. It means that τ_{m+1} is released after the upper bound on the finish time of all of the higher priority tasks, and hence, as soon as it is released, it will be the highest priority task in the system and must be scheduled by RM. Putting all of the results together, we have $s_{i,m+1} = \max\{A_{i,m+1}, f_{i,m}\}$ which proves (21). Since none of the high priority tasks will interfere with τ_{m+1} after $s_{i,m+1}$, the upper bound on its finish time will be the minimum between T_i and $C_{m+1} + s_{i,m+1}$. This proves (22). Moreover, since there is no interference in the execution of τ_{m+1} from the higher priority tasks after $s_{i,m+1}$, the lower bound of its effective workload will be $f_{i,m+1} - s_{i,m+1}$ which completes the proof. □

Using Lemma 3 we can obtain the lower bound of the

effective workload of all of the high priority tasks as

$$\omega_i = \sum_{1 \le j < i} \big(f_{i,j} - s_{i,j}\big) \tag{24}$$

THEOREM 2. *Given task set τ can be scheduled by RM only if the following condition holds*

$$T_i \ge \sum_{1 \le j \le i} \Big\lfloor \frac{T_i}{T_j} \Big\rfloor C_j + \sum_{1 \le j < i} \big(f_{i,j} - s_{i,j}\big) \tag{25}$$

where $s_{i,j}$ and $f_{i,j}$ are obtained from (21) *and* (22)*, respectively.*

PROOF. As shown in Lemma 1, (18) is the workload of higher priority jobs (including τ_i) which their absolute deadline is smaller than or equal to T_i in the interval $[0, T_i]$. Also according to Lemma 3, (24) is the lower bound on the effective workload of the high priority jobs with release time smaller than T_i and absolute deadline larger than T_i. Consequently, (25) is the lower bound on the effective workload of higher priority jobs together with τ_i, which must be finished within the interval $[0, T_i]$. If the lower bound of the workload, i.e., Ω_i, is larger than the duration of the interval, i.e., T_i, then τ_i must give its chances to one of the higher priority jobs which are released before T_i, and hence, cannot be finish. □

In the next step, we convert the necessary schedulability test in Theorem 2 to another necessary test which is a function of the utilization of the tasks and their period ratio.

THEOREM 3. *Given a task set τ, the following inequality is a necessary schedulability condition for RM*

$$\sum_{1 \le j \le i} u_j \le 1 + \sum_{1 \le j \le i} \frac{\gamma_{i,j}}{k_{i,j} + \gamma_{i,j}} u_j - \frac{\min\{u_1, \gamma_{i,1}\}}{k_{i,1} + \gamma_{i,1}} \tag{26}$$

PROOF. According to Lemma 2, equation (20) is the exact workload of the last job of τ_1 before T_i. If $\min\{C_1, T_i - A_{i,1}\} < C_1$, then we have $f_{i,1} = T_i$, which means that according to our formulation, the safe upper bound on the start time of other high priority tasks will be T_i, leading to $f_{i,j} = s_{i,j} = T_i$, $\forall j, 1 < j < i$. Otherwise if $\min\{C_1, T_i - A_{i,1}\} = C_1$, we know that τ_1 must be completely considered in ω_i. However, in the worst-case we might have $f_{i,1} = T_i$ which means that there might be no other chance for other high priority jobs to add a portion of their workload to ω_i. Consequently, in the worst-case, $\omega_i = f_{i,1} - s_{i,1} = \min\{C_1, T_i - A_{i,1}\}$. We use (25) which is a necessary schedulability test, remove $\sum \omega_i$ and replace it with $f_{i,1} - s_{i,1}$ which can further be simplified as

$$T_i \ge \sum_{1 \le j \le i} \Big\lfloor \frac{T_i}{T_j} \Big\rfloor C_j + \min\Big\{C_1, T_i - \Big\lfloor \frac{T_i}{T_1} \Big\rfloor T_1\Big\} \tag{27}$$

We divide (27) by T_i and replacing $\lfloor T_i/T_j \rfloor$ by $k_{i,j}$, thus

$$1 \ge \sum_{1 \le j \le i} \frac{k_{i,j} C_j}{T_i} + \min\Big\{\frac{C_1}{T_i}, 1 - \frac{k_{i,1} T_1}{T_i}\Big\} \tag{28}$$

From (1), we have $T_1/T_i = 1/(k_{i,1} + \gamma_{i,1})$, thus

$$1 - \frac{k_{i,1} T_1}{T_i} = 1 - \frac{k_{i,1}}{k_{i,1} + \gamma_{i,1}} = \frac{\gamma_{i,1}}{k_{i,1} + \gamma_{i,1}}$$

Also we have

$$\frac{C_1}{T_i} = \frac{T_1 u_1}{T_i} = \frac{u_1}{k_{i,1} + \gamma_{i,1}}$$

Now we put all of these simplifications back in (28)

$$1 \ge \sum_{1 \le j \le i} \frac{k_{i,j}}{k_{i,j} + \gamma_{i,j}} u_j + \frac{\min\{u_1, \gamma_{i,1}\}}{k_{i,1} + \gamma_{i,1}} \tag{29}$$

Moreover, we can simplify $\sum \frac{k_{i,j}}{k_{i,j}+\gamma_{i,j}} u_j$ as

$$\sum_{1 \le j \le i} \frac{k_{i,j}}{k_{i,j} + \gamma_{i,j}} u_j = \sum_{1 \le j \le i} u_j - \sum_{1 \le j \le i} \frac{\gamma_{i,j}}{k_{i,j} + \gamma_{i,j}} u_j \tag{30}$$

Using (30) we rewrite (29) to obtain (26), and hence, the claim is proven. □

One of the properties of the necessary condition in Theorem 3 is that this test becomes tighter than $U \le 1$ if

$$\sum_{1 \le j \le i} \frac{\gamma_{i,j}}{k_{i,j} + \gamma_{i,j}} u_j < \frac{\min\{u_1, \gamma_{i,1}\}}{k_{i,1} + \gamma_{i,1}} \tag{31}$$

because then the right-hand-side of (26) becomes smaller than 1, which means that $U \le 1 - a$ where $a \in \mathbb{R}^{>0}$.

It is worth noting that if it was possible to calculate the exact amount of the effective workload of the high priority jobs which are released before T_i and have absolute deadline after T_i, it would be possible to derive an exact schedulability test which is based on the workload calculation in the interval from 0 to T_i instead of calculating the worst-case response time of the tasks. As one of the contributions of this paper, in the next theorem we derive a condition in which the necessary schedulability test in Theorem 2 becomes a sufficient condition too, meaning that we provide a polynomial-time exact schedulability test for RM in those specific conditions.

THEOREM 4. *If the following condition holds, the task set τ defined in Sect. 2 (which is either sporadic or periodic with synchronous release) is schedulable by RM if and only if* (25) *holds for every task τ_i, $1 \le i \le n$.*

$$A_{i,1} \le A_{i,2} \le \ldots \le A_{i,i-1} \tag{32}$$

PROOF. According to (32), the order of the release of the last jobs of the high priority tasks must be the same as the order of their priorities. With this condition, the last job of τ_1 is executed in the interval $[s_{i,1}, f_{i,1}]$. Since $A_{i,2} > A_{i,1}$, the last job of τ_2 comes either before $f_{i,1}$ or after. In the former case, it will be the highest priority task at $f_{i,1}$ and can be scheduled in the interval $[f_{i,1}, f_{i,1} + C_2]$. Since at its release time, the processor is busy by executing τ_1, there is no other chance for τ_2 to start earlier than $f_{i,1}$. In the latter case where $A_{i,2} > f_{i,1}$, the task τ_2 will be the highest priority task in the system right at its release since the last instance of τ_1 has already been executed and it will not be released again in the system (due to the definition of $A_{i,1}$). As a result, the upper bound of the start time of τ_2 which is calculated in (21) is equal to the exact start time of τ_2. The same rationale can be applied to τ_3 and other high priority tasks, one by one. Consequently, ω_i becomes the exact effective workload of the higher priority jobs with deadline larger than T_i. It means that if (25) holds, there exists a

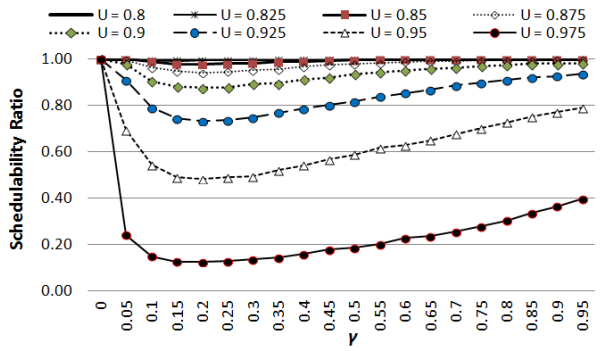

Figure 6: Effect of period residual on schedulability.

time such as $t \in [0, T_i]$ when the processor has nothing to schedule, including τ_i. Although we do not know the exact value of t, we know it exists since (25) holds. As a result, the worst-case response time of τ_i happens before or at T_i, which means that τ_i is schedulable [2]. □

6. EXPERIMENTAL RESULTS

In the first experiment, we study the effect of period ratio on the schedulability of RM. We measure the schedulability ratio, which is the ratio of schedulable task sets to the total number of generated task sets. For Fig. 1, 2-(a), and 3-(a) we have used the exact schedulability tests while in Fig. 7 we show the performance of different schedulability tests such as DCT [12], Park [18], linear method, our test in (11) which is based on verifying $t = T_i$ in (3), and finally, our necessary schedulability test in (25). DCT algorithm finds the largest set of harmonic periods which are smaller than the given periods. Then it checks whether using these periods the utilization is still smaller than 1 or not. In the linear method, (3) is used while $\lceil t/T_j \rceil$ is approximated by $(t/T_j) + 1$. Test of Park is based on verifying (3) at $t = \{A_{i,j}\}_{1 \le j \le i}$.

Effect of the minimum and maximum CPR on the schedulability of RM has been shown in Fig. 1 and the effect of period residual has been shown in Fig. 6. For these experiments, we have generated random task sets with 10 tasks. Each data point reports average schedulability ratio of 20,000 randomly generated task sets. To generate those task sets, we have used uUniFast [4] and generated random utilization values with total sum denoted in the diagram. Each period has been generated as $T_i = K_{i,i-1} \times T_{i-1}$ where T_1 has been selected with random distribution from $[1, 10]$. In Fig. 1-(a) and (b), $K_{i,i-1}$ is a random value with uniform distribution in range $[1, K^{max}]$ and $[K^{min}, 4]$, respectively. In Fig. 6, $T_i = (k_{i,i-1} + \gamma_{i,i-1}) \times T_{i-1}$ where $k_{i,i-1} \in \{1, 2, 3\}$ and $\gamma_{i,i-1}$ is the horizontal axis. After generating periods, we have assigned the execution times as $C_i = u_i T_i$.

As shown in Fig. 1-(a), with the increase in the consecutive period ratio, schedulability ratio increases. According to (11), by the increase in u_i, the effect of $\alpha_{i,j}$ (see (12)) becomes more visible as we can see it in the schedulability ratio of task sets with 0.975 utilization. Fig. 1-(b) shows that with the growth of K^{min} which implies larger CPR, the schedulability significantly increases. The reason for the drops in the schedulability around $K^{min} = 2$ and 3 is that K^{min} affects $K_{i,j}$ which in turn affects the the schedulability. We have seen that when $K^{min} = 2$ or 3, the average value of $\gamma_{i,i-1}$ is around 0.5 which is not very helpful for a task set with $U = 0.975$. We have seen that the drops of the schedulability ratio are matched with the drops in the average value of consecutive residuals.

[2]A more detailed proof is available in the Appendix

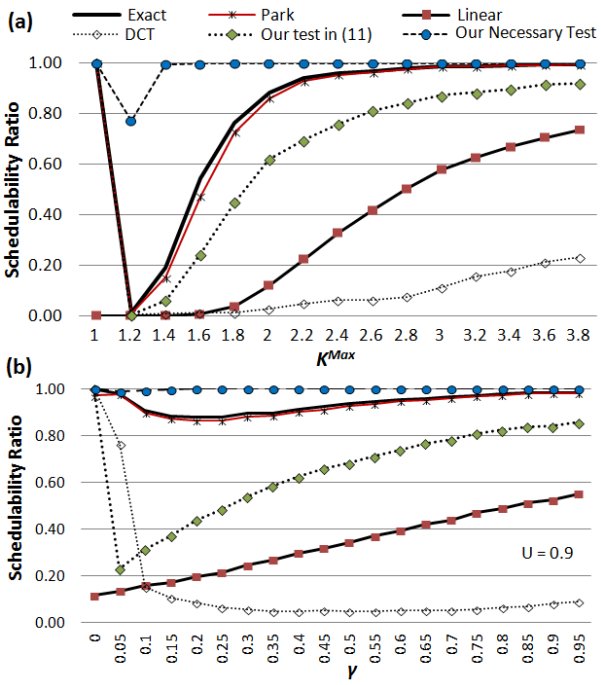

Figure 7: Schedulability ratio based on different schedulability tests as a function of K^{max} and period residual.

As depicted in Fig. 6, for example in $U \le 0.9$, if we have period residual larger than 0.5, the chance that the task set becomes schedulable is high. According to this diagram, to have a higher schedulability it is better to have larger residual if the system is not highly utilized, e.g., $U < 0.95$. Otherwise, it would be better to have very small period ratio.

Next, we evaluate performance of our necessary test with the same task set generation which we used for Fig. 1-(a) and Fig. 6 for $U = 0.9$. As shown in Fig. 7-(a), the test of Park [18] is almost as good as the exact test. Also in most cases, our test in (11) which only considers $t = T_i$ in evaluating (3) is much more efficient than the linear test. When consecutive period ratio is small our necessary test is able to detect 12% of non-schedulable task sets, e.g., in $K^{max} < 1.4$. Our necessary test is more efficient if it is used for task sets with small CPR.

7. MORE EXPERIMENTS

As suggested by Lehoczky in [15], we can use a fixed point method to obtain the WCRT of each task. It is suggested to start from $R_i^{(0)} = C_i$ and in each step, use the previously calculated $R_i^{(m-1)}$ to obtain $R_i^{(m)}$ as follows:

$$R_i^{(m)} \ge C_i + \sum_{1 \le j < i} \left\lceil \frac{R_i^{(m-1)}}{T_j} \right\rceil C_j \qquad (33)$$

until $R_i^{(m)} = R_i^{(m-1)}$.

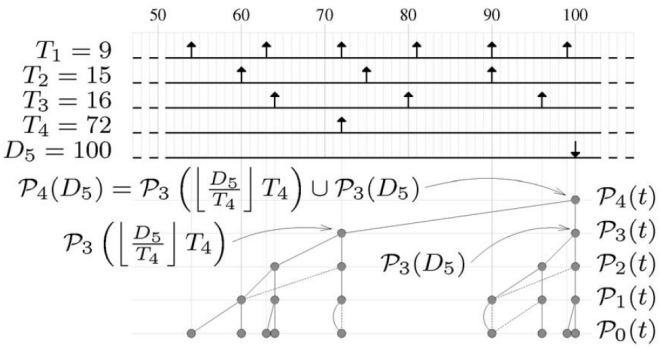

Figure 8: An example from [3] to show how the reduced set of points is calculated for verifying the schedulability of τ_5.

Later in [3], another method has been presented to calculate the WCRT of a task by finding a reduced set of points where (3) must be evaluated. This set is obtained from the following recursive function

$$\mathcal{TS}(\tau_i) \doteq \mathcal{P}_{i-1}(D_i) \tag{34}$$

$$\mathcal{P}_j(t) = \begin{cases} \{t\} & j = 0 \\ \mathcal{P}_{j-1}\left(\left\lfloor \frac{t}{T_j} \right\rfloor T_j\right) \cup \mathcal{P}_{j-1}(t) & \text{otherwise} \end{cases} \tag{35}$$

where the starting point of the recursive function is $\mathcal{P}_{i-1}(D_i)$. This function recursively calls itself until it reaches to the leaves, i.e., $P_0(t) = t$, which in fact is one of the candidate points in the final output. If two leaves in the search tree are equal, only one of them is kept in the resulting set. An example has been shown in Fig. 8 (Fig. 3 in [3]). This method of obtaining WCRT is called hyper-plane exact test (HET).

The computational complexity of HET is said to be independent from the periods or period ratios. However, in 2008, Davis et al. [8] have shown that in practice, the number of ceiling operations needed to perform the WCRT test using HET does not scale if periods are selected from wider ranges and can have wider variety. We have shown a part of these results in Fig. 9 (Table 5 from [8]). This table shows the number of ceiling operations that are needed to verify (3) until the WCRT is found. This number is shown as a function of the order of magnitude of the growth in the range from which periods are selected. In this experiments, each task set has 24 tasks with an overall utilization of 85 percent. Periods are generated using a uniform distribution from $[1, 10^x]$ where x is the order-of-magnitude mentioned in the table.

In the next set of experiments we evaluate total number of operations which has been done by each of those two algorithms (i.e., RTA and HET). Note that instead of implementing (34), we directly use the test which is presented in [3] (the FPTest algorithm). The number which we report is based on counting all operations that can be performed in $O(1)$, as 1. For example, if we have a loop that iterates 10 times and in each iteration it performs 5 non-loop simple instructions, then we report 10 operations. Consequently, each call to the WorkLoad algorithm in [3] is considered as only 1 operation.

In order to generate random periods, here we use the method suggested by [10] which implements a log-uniform

	Orders of magnitude spanning task periods					
Algorithm	**1**	**2**	**3**	**4**	**5**	**6**
RTA	1247	1652	2050	2462	2847	3297
HET	380	1326	5642	23014	81636	197642
HET/RTA	0.3	0.8	2.75	9.35	28.7	60.0

Figure 9: (Table 5 from [8]): a comparison between HET and RTA in terms of the number of ceiling operations required for verifying schedulability. The header of the table shows how wide are the ranges from which periods are randomly selected (with a uniform distribution).

distribution for periods. First we generate a random value r_i with uniform distribution as:

$$r_i \approx U(log(T_{min}), log(T_{max} + T_g)) \tag{36}$$

where T_g is the granularity of the periods, i.e., all periods will be integer multiples of T_g. In our case, we assume $T_g = 1$. Each period is obtained as

$$T_i = \left\lfloor \frac{exp(r_i)}{T_g} T_g \right\rfloor \tag{37}$$

Figures 10 to 13 show the results of experiments using log-uniform periods. For each data point, 20000 random task sets with $U = 0.9$ are generated. Utilization of each task is chosen randomly using uUniFast algorithm. Similar to Fig. 3 and Fig. 2, here we have shown how different period generation method affects the average, minimum, and maximum CPR. Also, for the sake of clarity, we have plotted the minimum and maximum utilization of the tasks in each task set in part (c) of the diagrams. Note that since we have used uUniFast to generate the utilizations, in all four figures part (c) is the same. This diagram shows that if uUniFast is used to generate random utilization, it will probably not generate a case where 1 task has a large utilization, e.g., 80% and the others have together 10% utilization. In fact, *one possible conclusion here is that uUniFast helps RM to have better results.* Equation (11) shows that if u_j values (task utilization values) are small, they can very well hide the negative effect of $k_{i,j}$ and $\gamma_{i,j}$ on the schedulability. Our results emphasis on the importance of *revisiting* the existing random task set generation methods and their effects on RM schedulability (or different schedulability tests).

In part (d) of Figures 10 to 13, we have shown the average number of operations needed to analyze the schedulability of a task set using RTA and HET algorithms. In all of the diagrams, the horizontal axis is the number of tasks.

As it can be seen, HET is not scale-able when the number of tasks increase and periods are fairly different from each other. On the other hand, RTA method is very efficient even in those situations.

8. RELATED WORK

One of the pioneering works to characterize RM-friendly task sets is [12] where it has been shown that RM is able to schedule a harmonic task set up to 100% utilization. Our schedulability test in (16) covers the case of fully harmonic tasks and brings another proof for their schedulability. An algorithm to derive the minimum number of harmonic chains has been presented in [14]. In addition, it has been shown

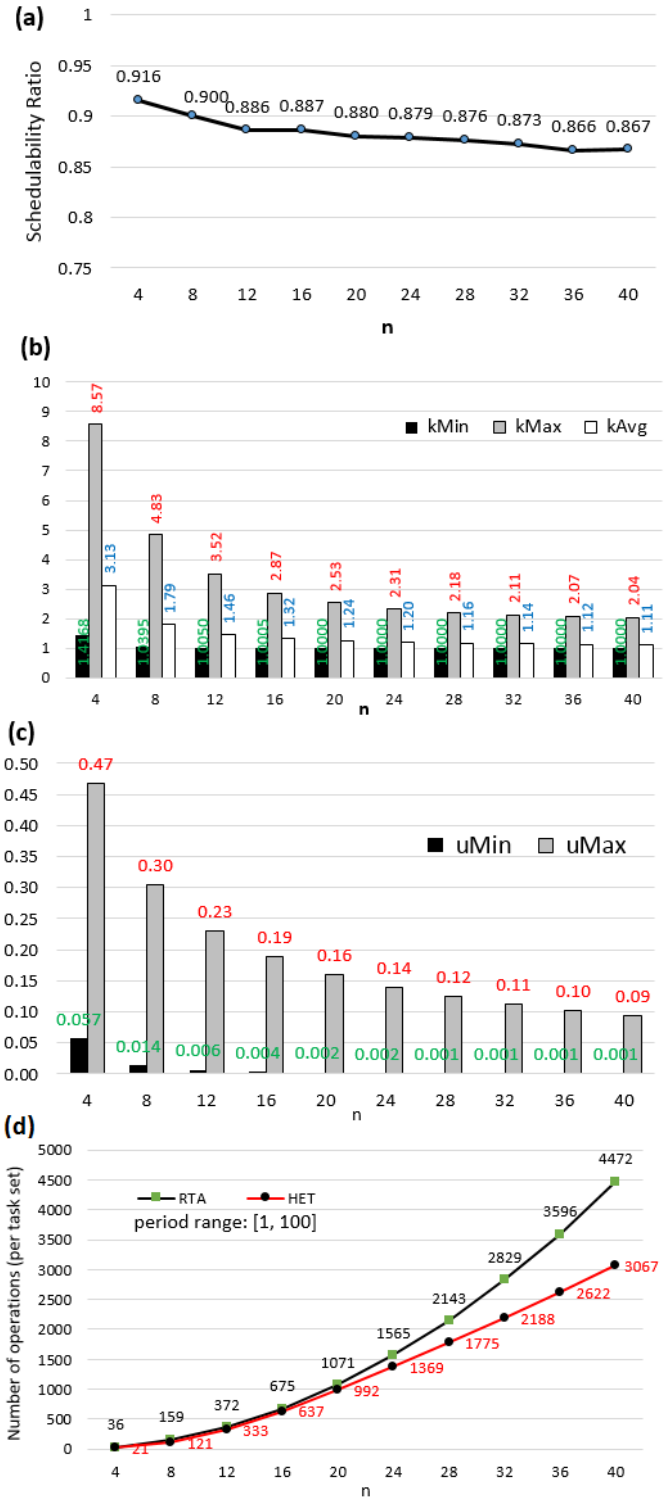

Figure 10: Periods with log-uniform distribution from range [1, 100].

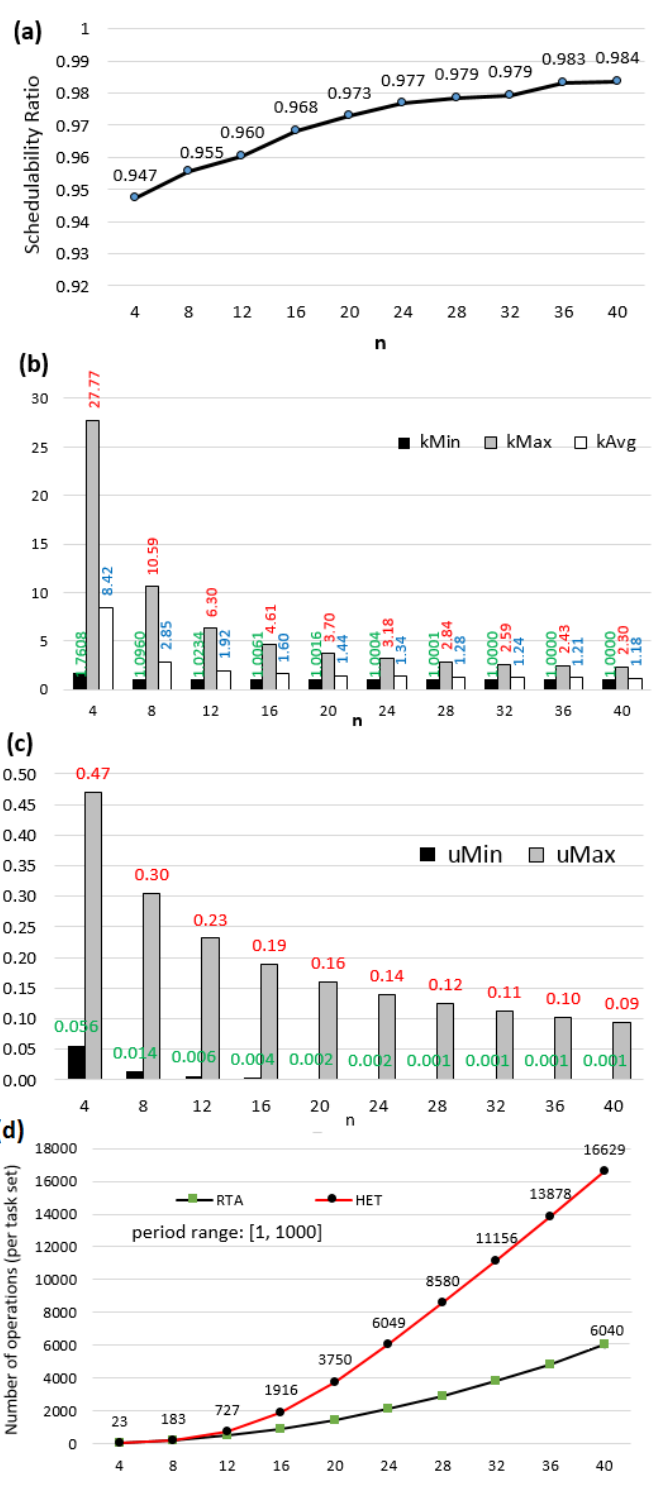

Figure 11: Periods with log-uniform distribution from range [1, 1000].

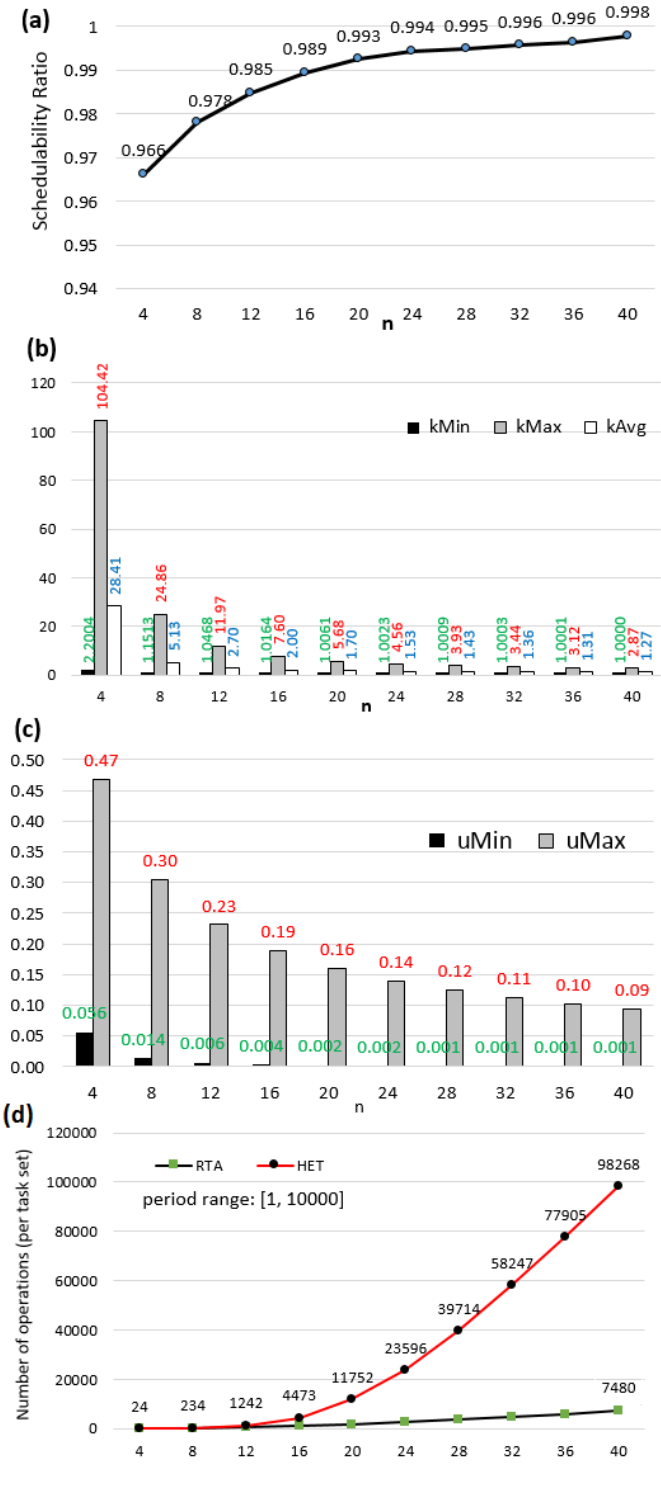

Figure 12: Periods with log-uniform distribution from range [1, 10000].

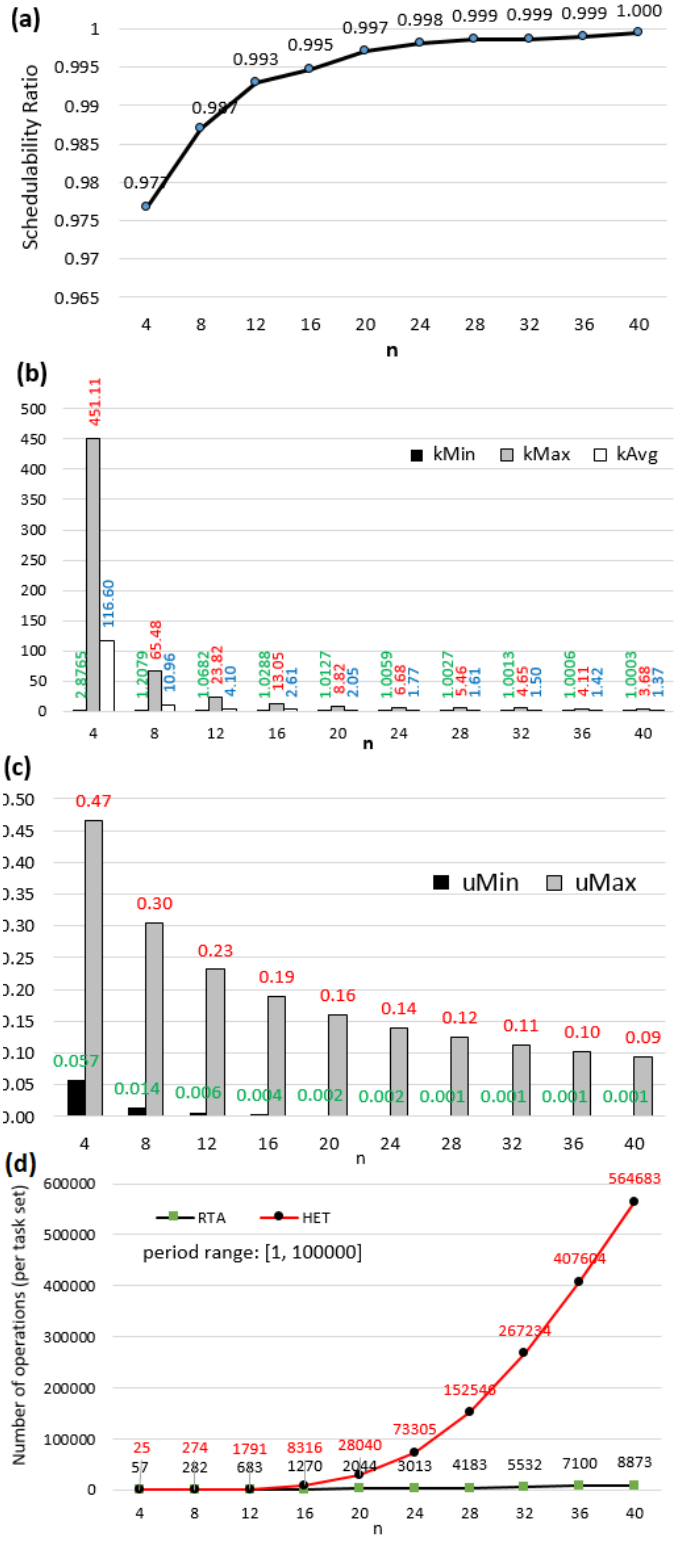

Figure 13: Periods with log-uniform distribution from range [1, 100000].

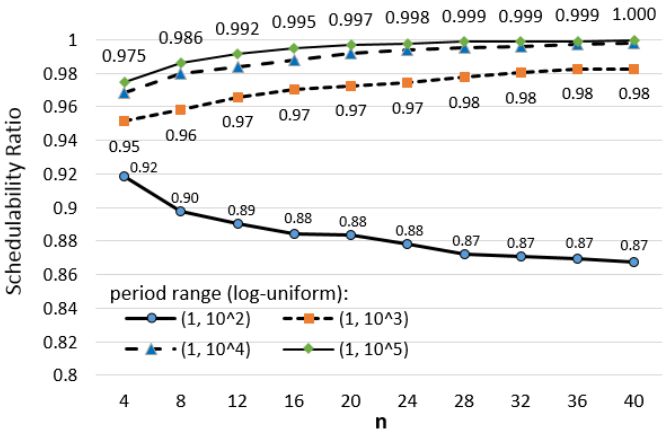

Figure 14: Comparing schedulability ratio in Fig. 10 to 13 together.

that each harmonic chain can be considered as one aggregating task in the test, and hence, it leads to a reduced number of tasks in the schedulability analysis process. In our work, 1) we *quantify* the effect of period residual (which shows how periods are close to harmonic periods) on the schedulability, and 2) we do not need to group the tasks into harmonic chains in order to reduce the complexity of the test.

The effect of the tasks parameters on RM schedulability has been studied in several previous works. A numerical evaluation has been performed in [10] showing that selecting periods from larger ranges, e.g., using *log-uniform* distribution, will increase RM schedulability. In our work, we have quantified this effect. In [15] a stochastic analysis has been performed to show that when the period ratio approaches to infinity, the maximum schedulable utilization reaches to 1. Also, the asymptotic value of this utilization depends only on the period distributions (that is, it does not depend on the distribution of WCETs). Both of these results are motivations to our work to explicitly consider the role of period ratios on the utilization bound of the RM. Meanwhile, in the current work, instead of the size of the task set (and a stochastic situation), we focus on the relative value of the periods.

A sensitivity analysis of RM schedulability with respect to the WCET and period was done by Bini et al. [5]. In our work, we consider the mutual relation between periods rather than periods themselves. In comparison, we take into account the fact that changing the value of a period irrespective of its ratio to the other periods is not necessarily a good choice for sensitivity analysis. In [8, 10], it was noticed (through experimental results) that having periods with different *order of magnitudes* is an important factor when generating random tasks. Still, they did not analytically studied the effect of period ratio on the schedulability.

In [19] an efficient schedulability bound for RM has been presented based on the ratio between the smallest and the largest periods and utilization of the tasks. Our work considers more information about relative period ratio and residual of any pair of tasks in the tests. Moreover, our main goal is to use the tests to understand the effect of period ratio on RM schedulability. In [2], a utilization based test is introduced for arbitrary deadline tasks. This test uses information about the minimum ratio between any two consecutive periods, i.e., K^{min}. Different from them, we exploit information about relative period ratio of different tasks, which leads to a less pessimistic test. The analysis in [2] becomes very pessimistic for tasks with almost similar periods (small period residual), while our results show that in those cases, schedulability increases. We will consider performing a quantitative evaluation between our approach and theirs after extending our method to arbitrary deadline tasks.

In [7] it has been shown that an RM-schedulable task set with $U = 1$ will only be schedulable if the task set is semi-harmonic, i.e., $T_n|T_i; \forall i$. Our result in (16) provides a more general way to identify schedulable tasks with small, large, or zero $\gamma_{i,j}$. Besides, we show how using $T_i|T_j; \forall 1 \leq j < i$, or having condition (32), the exact schedulability analysis can be done in polynomial-time for some of the tasks.

9. CONCLUSIONS

In this paper, we have quantified the effect of period ratio on the schedulability of RM in order to characterize RM-friendly task sets (which can be scheduled by RM with a high chance). To quantify this effect, we have introduced sufficient schedulability tests which describe the schedulability as a function of task utilizations and relative period ratio. We have shown that not only harmonic periods but also the periods that are very close to an integer multiple of the smaller periods are RM-friendly. Moreover, when tasks have large relative period ratio, RM decisions become similar to EDF decisions, which finally leads to high schedulability ratio for those task sets. We have introduced a necessary schedulability test for RM, and then showed that if a certain condition holds for a task, our necessary (yet polynomial-time) test becomes an exact schedulability test. As a future work, we will consider constrained and arbitrary deadline tasks with release jitter and then use our results to build more efficient partitioning algorithms for multiprocessor systems.

Acknowledgment

We would like to thank our anonymous reviewers for their insightful comments. This work has been supported by Alexander von Humboldt Foundation.

10. REFERENCES

[1] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[2] E. Bini. The quadratic utilization upper bound for arbitrary deadline real-time tasks. *IEEE Transactions on Computers*, 64(2):593–599, 2015.

[3] E. Bini and G. Buttazzo. Schedulability analysis of periodic fixed priority sysyems. *IEEE Transactions on Computers*, 53(11):1462–1473, 2004.

[4] E. Bini and G. Buttazzo. Measuring the Performance of Schedulability Tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[5] E. Bini, M. Di Natale, and G. Buttazzo. Sensitivity analysis for fixed-priority real-time systems. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 10–22, 2006.

[6] E. Bini, A. Parri, and G. Dossena. A Quadratic-Time Response Time Upper Bound with a Tightness

Property. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 13–22, 2015.

[7] J. Chen. *Extensions to fixed priority with preemption threshold and reservation-based scheduling.* PhD thesis, University of Waterloo, Waterloo, Canada, 2005.

[8] R. Davis, A. Zabos, and A. Burns. Efficient Exact Schedulability Tests for Fixed Priority Real-Time Systems. *IEEE Transactions on Computers*, 57(9):1261–1276, 2008.

[9] A. Diaz-Ramirez, P. Mejia-Alvarez, and L. E. Leyva-del Foyo. Comprehensive Comparison of Schedulability Tests for Uniprocessor Rate-Monotonic Scheduling. *Journal of Applied Research and Technology*, 11(3):408–436, 2010.

[10] P. Emberson, R. Stafford, and R. Davis. Techniques For The Synthesis Of Multiprocessor Tasksets. In *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, pages 6–11, 2010.

[11] N. Fisher and F. Dewan. Approximate Bandwidth Allocation for Compositional Real-Time Systems. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 87–96, 2009.

[12] C.-C. Han and H.-Y. Tyan. A Better Polynomial-time Schedulability Test for Real-time Fixed-priority Scheduling Algorithms. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 36–45, 1997.

[13] S. Kramer, D. Ziegenbein, and A. Hamann. Real world automotive benchmark for free. In *International Workshop on Analysis Toolsand Methodologies for Embedded Real-time Systems (WATERS)*, 2015.

[14] T.-W. Kuo and A. Mok. Load adjustment in adaptive real-time systems. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 160–170, 1991.

[15] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior, 1989.

[16] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of ACM*, 20(1):46–61, 1973.

[17] M. Nasri and G. Fohler. An Efficient Method for Assigning Harmonic Periods to Hard Real-Time Tasks with Period Ranges. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 149–159, 2015.

[18] M. Park and H. Park. An efficient test method for rate monotonic schedulability. *IEEE Transactions on Computers*, 63(5):1309–1315, 2014.

[19] H.-W. Wei, K.-J. Lin, W.-C. Lu, and W.-K. Shih. Generalized rate monotonic schedulability bounds using relative period ratios. *Information Processing Letters*, 107(5):142–148, 2008.

Appendix

A more detailed proof for Theorem 4

PROOF. For the proof, we need to show that (25) with (32) provides a sufficient schedulability condition. As mentioned in Sect. 3, existing a time instant $t \leq T_i$ which satisfies (3) implies schedulability of task τ_i. As a result, we need only to show that, when (32) holds, then (25) leads to the existence of such a time instant t.

We first review a set of properties used in the proof:

$$\lfloor x \rfloor + 1 \geq \lceil x \rceil, \qquad \forall x \in \mathbb{R}, \tag{38a}$$
$$x \leq n \Rightarrow \lceil x \rceil \leq n, \qquad \forall x \in \mathbb{R}, \forall n \in \mathbb{Z}, \tag{38b}$$
$$n \leq x \Rightarrow n \leq \lfloor x \rfloor, \qquad \forall x \in \mathbb{R}, \forall n \in \mathbb{Z}, \tag{38c}$$
$$f_{i,j} \leq T_i, \qquad 1 \leq j \leq i-1, \tag{38d}$$
$$f_{i,j-1} \leq f_{i,j}, \qquad 2 \leq j \leq i-1, \tag{38e}$$
$$f_{i,j} < T_i \Rightarrow f_{i,j} - s_{i,j} = C_j, \quad 2 \leq j \leq i-1. \tag{38f}$$

We elaborate the proof considering two possible cases.

a) $f_{i,i-1} < T_i$: According to (38e), this case implies that $f_{i,j} < T_i$ for all j; $1 \leq j \leq i-1$. Consequently, due to (38f), we have $\sum_{1 \leq j < i} (f_{i,j} - s_{i,j}) = \sum_{1 \leq j < i} C_j$. Based on this relation, (25) can be written as

$$\begin{aligned} T_i &\geq \sum_{1 \leq j \leq i} \left\lfloor \frac{T_i}{T_j} \right\rfloor C_j + \sum_{1 \leq j < i} C_j \\ &= C_i + \sum_{1 \leq j < i} \left(\left\lfloor \frac{T_i}{T_j} \right\rfloor + 1 \right) C_j \end{aligned} \tag{39}$$

Regarding (38a), Equation (39) leads to

$$T_i \geq C_i + \sum_{1 \leq j < i} \left\lceil \frac{T_i}{T_j} \right\rceil C_j \tag{40}$$

Comparing this equation with (3) reveals that we have found a time instant $t \leq T_i$ which satisfies (3).

b) $f_{i,i-1} = T_i$: First, let define a parameter t_b as

$$t_b = \min\left(S \cup \{s_{i,i-1}\}\right) \tag{41}$$

where $S = \{s_{i,j} \mid 1 \leq j < i-1 \wedge s_{i,k} = f_{i,k-1} \text{ for all } k \in [j+1, \ldots, i-1]\}$. Intuitively, t_b denotes the beginning of the maximum interval which ends in T_i and the processor is continually busy with running higher priority tasks (than τ_i) during that interval. Further, let m be the respective task index, i.e.

$$m = \arg\min_{1 \leq j < i} \{s_{i,j} \mid s_{i,j} = t_b\}. \tag{42}$$

Based on Lemma 5, we have $f_{i,j} < T_i$ for $j < m$, which, according to (38f), implies $f_{i,j} - s_{i,j} = C_j$. As a result, we can write

$$\sum_{1 \leq j < m} (f_{i,j} - s_{i,j}) = \sum_{1 \leq j < m} C_j. \tag{43}$$

Moreover, based on Lemma 6, we have

$$\sum_{m \leq j < i} (f_{i,j} - s_{i,j}) = T_i - A_{i,m}. \tag{44}$$

Using (43) and (44) we can rewrite (25) as

$$T_i \geq \sum_{1 \leq j \leq i} \left\lfloor \frac{T_i}{T_j} \right\rfloor C_j + \sum_{1 \leq j < m} C_j + T_i - A_{i,m} \tag{45}$$

which yields

$$\begin{aligned} A_{i,m} &\geq \sum_{1 \leq j \leq i} \left\lfloor \frac{T_i}{T_j} \right\rfloor C_j + \sum_{1 \leq j < m} C_j \\ &= C_i + \sum_{1 \leq j < m} \left(\left\lfloor \frac{T_i}{T_j} \right\rfloor + 1 \right) C_j + \sum_{m \leq j < i} \left\lfloor \frac{T_i}{T_j} \right\rfloor C_j \end{aligned}$$

Then, regarding (38a), we obtain

$$A_{i,m} \geq C_i + \sum_{1 \leq j < m} \left\lceil \frac{T_i}{T_j} \right\rceil C_j + \sum_{m \leq j < i} \left\lfloor \frac{T_i}{T_j} \right\rfloor C_j \tag{46}$$

According to Lemma 7, we know that $\lfloor T_i/T_j \rfloor \geq \lceil A_{i,m}/T_j \rceil$ for $j \geq m$. Thus, from (46) we can obtain:

$$A_{i,m} \geq C_i + \sum_{1 \leq j < m} \left\lceil \frac{T_i}{T_j} \right\rceil C_j + \sum_{m \leq j < i} \left\lceil \frac{A_{i,m}}{T_j} \right\rceil C_j \qquad (47)$$

In addition, as $T_i \geq A_{i,m}$, we can further write

$$A_{i,m} \geq C_i + \sum_{1 \leq j < m} \left\lceil \frac{A_{i,m}}{T_j} \right\rceil C_j + \sum_{m \leq j < i} \left\lceil \frac{A_{i,m}}{T_j} \right\rceil C_j \qquad (48)$$

With (48), we have shown that (3) holds for some $t \leq T_i$ (i.e., for $t = A_{i,m}$).

Now, we present lemmas used in the proof of Theorem 4.

LEMMA 4. *Consider m as defined in* (42)*. Then, we have $s_{i,m} = A_{i,m}$.*

PROOF. If $m = 1$, then $s_{i,1} = A_{i,1}$ by definition. For $m > 1$, if $s_{i,m} \neq A_{i,m}$, then, according to (21), we must have $s_{i,m} = f_{i,m-1}$. As a result, regarding (41) and (42), m is not the minimum task index for which the condition in (42) holds, which contradicts its definition. □

LEMMA 5. *Considering m as defined in* (42)*, we have $f_{i,j} < T_i$ for $j < m$.*

PROOF. We prove the contrapositive of the lemma. If the lemma does not hold, then $\exists j < m$ such that $f_{i,j} = T_i$. This means that $s_{i,k} = f_{i,k} = T_i$ for all $k \in [j+1, \ldots, i-1\}$ which implies $s_{i,k} = f_{i,k-1}$ for all $k \in [j+1, \ldots, i-1\}$. As a result, m is not the minimum task index which is inconsistent with its definition. □

LEMMA 6. *For m as defined in* (42)*, if $f_{i,i-1} = T_i$, then*

$$\sum_{m \leq j < i} (f_{i,j} - s_{i,j}) = T_i - A_{i,m}. \qquad (49)$$

PROOF. First, we rephrase the left-hand-side of (49) as

$$\begin{aligned} \sum_{m \leq j < i} f_{i,j} - s_{i,j} &= f_{i,i-1} - s_{i,m} + \sum_{m \leq j < i-1} f_{i,j} - \sum_{m < j < i} s_{i,j} \\ &= T_i - s_{i,m} + \sum_{m < j < i} (f_{i,j-1} - s_{i,j}) \end{aligned}$$

According to the definition of m (see (42) and (41)), we have $s_{i,j} = f_{i,j-1}$ for all $j \in [m+1, \ldots, i-1]$; hence, the above equation implies $\sum_{m \leq j < i} (f_{i,j} - s_{i,j}) = T_i - s_{i,m}$. Replacing $s_{i,m}$ with $A_{i,m}$ (based on Lemma 4) completes the proof. □

LEMMA 7. *Let m denote the parameter defined in* (42)*; then, for $j \geq m$, we have $\lceil A_{i,m}/T_j \rceil \leq \lfloor T_i/T_j \rfloor$*

PROOF. For the proof, we notice, according to (32), that for $j \geq m$, we have $\frac{A_{i,m}}{T_j} \leq \frac{A_{i,j}}{T_j}$. Since $\frac{A_{i,j}}{T_j} \in \mathbb{N}$, we can use (38b) to conclude that

$$\left\lceil \frac{A_{i,m}}{T_j} \right\rceil \leq \frac{A_{i,j}}{T_j} \qquad (50)$$

Additionally, we consider that $\frac{A_{i,j}}{T_j} \leq \frac{T_i}{T_j}$ which, as $\frac{A_{i,j}}{T_j} \in \mathbb{N}$ and according to (38c), gives

$$\frac{A_{i,j}}{T_j} \leq \left\lfloor \frac{T_i}{T_j} \right\rfloor \qquad (51)$$

Finally, mixing (50) and (51) yields $\lceil A_{i,m}/T_j \rceil \leq \lfloor T_i/T_j \rfloor$. □