# Comprehensive Comparison of Schedulability Tests for Uniprocessor Rate-Monotonic Scheduling

***Abstract*—Schedulability conditions are used in real-time systems to verify the fulfillment of the temporal constraints of task sets. In this paper, a performance analysis is conducted for the best-known real-time schedulability conditions that can be used in *online* admission control on uni-processor systems executing under the Rate-Monotonic scheduling policy. Since Liu and Layland introduced the Rate-Monotonic scheduling algorithm, many research studies have been conducted on the schedulability analysis of real-time periodic task sets. However, in most of the cases, the performance of the proposed schedulability conditions were compared only against the Liu and Layland test and not against the remaining schedulability tests. The goal of this paper is to provide guidelines for system designers in order to decide which schedulability condition provides better performance under different task characteristics. Extensive simulation experiments were conducted to evaluate the inexact schedulability conditions and compare their performance and computational complexity.**

***Index Terms*—Real-Time Systems, Real-Time Task Scheduling, Rate-Monotonic, Schedulability Test, Uniprocessors**

## I. Introduction

In a real-time system, the *scheduling algorithm* decides an *order of execution* of the tasks and the *amount of time* allowed to each task in the system so that no task (for *hard* real-time systems), or a minimum number of tasks (for *soft* real-time systems), misses their deadlines. To verify if a scheduling policy guarantees the fulfillment of the temporal constraints of a task set, real-time systems designers use different exact or inexact *schedulability conditions* (also known as *schedulability tests*). The schedulability condition indicates if a given task set can be scheduled with a given scheduling algorithm such that no tasks in the set miss their deadlines. When a new task is created in a dynamic real-time system, an *online admission control* mechanism that uses a schedulability test, guarantees predictability if the new task is admitted. Examples of these kind of systems are those with Quality-of-Service (QoS) requirements, such as multimedia systems [23] [33], communication services [3][9], and automated flight control [1]. Other examples are found on the scheduling of real-time traffic over networks [30][10], or in open systems environments [19][20].

Exact schedulability tests usually have high time complexities and may not be adequate for online admission control if the system has a large amount of tasks or a dynamic workload. In contrast, most of the inexact schedulability tests provide low complexity *sufficient* schedulability tests, which are suitable for use in online admission control mechanisms to decide the acceptance of the newly arriving tasks in the system. If a task set does not satisfy a sufficient schedulability test, it is not known if the task set can be feasibly scheduled using a given scheduling policy. For this reason, it is important to determine which inexact schedulability test provides a better performance, given the specific task set parameters.

The *Rate-Monotonic* (RM) scheduling algorithm assigns priorities proportionally to the task activation rates. Many other scheduling algorithms have been proposed, such as the *Earliest Deadline First* (EDF) [26] that allows a better use of the computational resources. However, since RM introduces low-computational overhead, is simple to implement and is predictable, it is widely used on most real-time operating systems and is supported by most real-time systems standards.

Liu and Layland first introduced the Rate-Monotonic algorithm along with a sufficient schedulability test [26]. They introduced the concept of *achievable utilization factor* to derive a low complexity test that is used to determine the schedulability of independent, periodic and preemptable task sets executed on one processor. The schedulability test introduced by Liu and Layland for RM states that a task set will not miss any deadline if the utilization factor of the task set, defined as $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$, is not greater than $n(2^{\frac{1}{n}} - 1)$, where $C_i$ and $T_i$ are the computation requirement and period of the task $\tau_i$, respectively, and *n* is the number of tasks. Unfortunately, this condition fails to identify many schedulable task sets when the system is heavily loaded.

After Liu and Layland's seminal work, many researchers, motivated by the low overhead and simplicity of RM, developed new tests that improved the test proposed by them. The improvement on these new tests was due to the introduction of additional timing parameters in the schedulability analysis, and in some cases, also to the transformation of the task sets.

When comparing the inexact schedulability conditions, the problem of evaluating their performance with respect to either the pessimistic Liu and Layland test or the exact schedulability test, becomes an important issue. The effectiveness of the schedulability test is measured in terms of the *acceptance ratio*. The higher the acceptance ratio, the better the test, which means that more tasks sets are schedulable. When the ratio is equal to one, it means that the schedulability condition finds as many schedulable task sets as those found by the exact condition.

Comparing the inexact schedulability conditions, using a

rigorous analytical approach, is not easy because each schedulability condition considers different task set parameters. Furthermore, some of them are based on algorithms, that is, transforming the original task set into an equivalent one. Consequently, the acceptance ratio of a given test is affected by the characteristics of the task set parameters. For these reasons, our aim is to evaluate the performance of the schedulability conditions through extensive simulations.

In this paper, we survey the inexact schedulability conditions that can be used in online admission control when the system is comprised of periodic and preemptable real-time tasks, using the Rate-Monotonic scheduling algorithm. We analyze the best-known inexact schedulability conditions for one processor that exists in the literature, and conduct extensive simulation experiments to evaluate and compare their performance in terms of the acceptance ratio and the computational complexity. Based on the results provided by the experimental evaluations, we provide guidelines to help system designers to decide, given a particular task set parameters and load conditions, which schedulability tests provide better or worst performance, and how they compare with each other under these different characteristics.

To our knowledge, no previous comparative analysis of the RM schedulability conditions has been conducted for real-time scheduling on one processor. Most of the published schedulability conditions compared their performance only against the schedulability condition introduced by Liu and Layland, and just a few of them compared their performance against any other.

The rest of this document is organized as follows: In Sections 2 and 3, an overview of the real-time scheduling theory and schedulability analysis of real-time systems is introduced. In Section 4, the schedulability conditions for RM on one processor are introduced, and in Section 5, extensive simulation experiments conducted to test and compare the performance of the inexact schedulability conditions are described. Finally, the conclusions appear in Section 6.

## II. Real-time systems scheduling

A real-time system is composed of several concurrent activities that are normally implemented as tasks. To schedule these tasks, real-time operating systems use scheduling algorithms to decide the *order of execution* of the tasks and the *amount of time* assigned to each task.

Scheduling algorithms of general-purpose operating systems are *non-deterministic* because the correctness of the system does not depend on the order in which every task is executed. In these operating systems, the scheduler is intended to provide *optimal performance*, *optimal usage of resources*, and *fairness* in resource assignment. In contrast, in real-time operating systems, the scheduler must restrict the non-determinism associated with the concurrent system, and must provide the means to predict the worst-case temporal behavior of the task set.

A *real-time scheduling algorithm* provides an *ordering policy* for the execution of the tasks (as in the non-real-time scheduling algorithm). A given real-time scheduling algorithm may produce *feasible* or *infeasible* schedules. In a feasible schedule, every job for a given task set always completes by its deadline. In contrast, in an infeasible schedule, some jobs may miss a few of their deadlines. A set of jobs is *schedulable* according to a given scheduling algorithm if, when using the algorithm, the scheduler always produces a feasible schedule. The criterion used to measure the performance of the scheduling algorithms for real-time applications is their ability to find feasible schedules of the given application whenever such schedules exist. A hard real-time scheduling algorithm is *optimal* if, for any feasible task set, it always produces feasible schedules [27].

The scheduling algorithms can be classified as *static* and *dynamic*. In a *static scheduling algorithm*, all scheduling decisions are provided *a priori*. For a given set of timing constraints, a table is constructed indicating the starting and completion times of each task, such that, no task misses its deadline. This approach is highly predictable, but when the parameters of the tasks change, the table must be recomputed and the system restarted.

In *dynamic scheduling algorithms*, the scheduling decisions are taken at run-time based on the *priorities* of the tasks. These priority values are used to decide the execution order of the tasks. Priority values can be assigned *statically* or *dynamically*, depending on the dynamic scheduling algorithm. If static priorities are used, the priority of each task remains fixed during the complete execution of the system, whereas if dynamic priorities are used, the priority of a task is allowed to change at any moment.

As mentioned before, Liu and Layland [26] introduced the first real-time scheduling algorithms for a single processor (Rate-Monotonic and Earliest Deadline First), and developed their corresponding schedulability analysis. RM assigns the highest priority to the task with the smallest period, and EDF assigns priorities to the tasks considering the proximity of each instance of a task with its deadline, so that the task with the closest relative deadline receives the highest priority. Liu and Layland demonstrated that RM and EDF are optimal for fixed and dynamic priority algorithms, respectively.

### A. System Model

In this paper, we consider a real-time system composed of a set of *n* real-time tasks $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$ on one processor under Rate-Monotonic. A task is usually a thread or a process within an operating system. The parameters that define a task $\tau_i$ are: the *execution time* $C_i$, the *period* $T_i$, and the *deadline* $D_i$. We will consider that only periodic tasks can be executed in the system, and we will consider that $T_i = D_i$. Each periodic task $\tau_i$ is composed of an infinite sequence of jobs. The period $T_i$ of the periodic task $\tau_i$ is a fixed time interval between the release times of consecutive jobs in $\tau_i$. Its execution time $C_i$ is the *maximum execution time* of all the jobs in $\tau_i$. The period and execution time of the task $\tau_i$ satisfies that $T_i > 0$ and $0 < C_i \leq T_i = D_i$, ($i$ = 1,..., $n$). The *utilization factor* of the task $\tau_i$ is defined as $u_i = \frac{C_i}{T_i}$ . The utilization factor of the task set, denoted as $U$, is the sum of the utilization of the tasks in the set, that is, $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$. We will consider that a job in $\tau_i$ that is released at time $t$, must complete within $D_i$, that

is, it must complete within the time interval ($t$, $t$+$D_i$]; where $D_i$ is the *relative deadline* of the task $\tau_i$. The release time of the first job in each task $\tau_i$ is called the phase of $\tau_i$, and is denoted as $\theta_i$.

We use $H$ to denote the least common multiple of $T_i$, for $i$ = *1, 2,..., n*. A time interval of length $H$ is called the *hyperperiod* of the task set.

In the model used in this paper, the following restrictions also apply:

- A1 The tasks are *independent*. That is, the arrival of a job of task $\tau_i$ is not affected by the arrival of any job of the other task $\tau_{j\neq i}$.
- A2 It is assumed that all tasks in the system can be *preempted* at any time.
- A3 The cost of the context switch of the tasks is considered negligible.
- A4 No resources, other than the CPU are shared among the tasks.

## III. Schedulability tests

A *schedulability test* defines a mathematical condition that is used to verify whether the task set meets its temporal restrictions for a given scheduling algorithm. The inputs of the test are the temporal parameters of the task set.

A test is said to be *sufficient* in the sense that a task set is schedulable if it satisfies the test. However, if the task set does not satisfy the sufficient test, it is not known whether the task set can be schedulable using that scheduling algorithm. A test is said to be *necessary* if all schedulable task sets satisfy the test. Otherwise, if a given task set satisfies the test, we cannot say that it is schedulable. Exact tests provide a *necessary* and *sufficient* condition. The *inexact* schedulability tests provide only a *sufficient* (but not necessary) schedulability condition.

Schedulability tests depend on the scheduling algorithm chosen and the knowledge of the parameters of the task set. The schedulability test in dynamic scheduling algorithms can be performed *off-line* or *online*. If the test is executed *off-line*, there must be complete knowledge of the set of tasks that are to be executed in the system along with the timing constraints imposed on every task (e.g., deadlines, precedence restrictions, execution times) before the execution of the system. In this case, the arrival of new tasks is not allowed while the system is executing, and the tasks cannot change their timing constraints.

In contrast, if the scheduling test is performed *online*, new arrivals are allowed at any time and the tasks can change their timing constraints during the execution of the system. In this test, the scheduler decides dynamically, by means of an *admission control mechanism*, if the acceptance of these new tasks will not cause other tasks to miss their deadlines.

The utilization bound $\hat{U}$, for a given real time scheduling algorithm, is the value such that any task set, whose utilization factor is no larger than $\hat{U}$, is schedulable under that scheduling algorithm. *Utilization-based* schedulability conditions verify if the utilization of the task set does not exceed the utilization bound (that is, $U \leq \hat{U}$).

We classify the inexact tests in accordance with the parameters used as follows:

- *Non-period-aware schedulability conditions*. These schedulability conditions derive the utilization bound using information about the *number of tasks* or the *utilization of the tasks* in the system. The tests based on the utilization found in the literature are:
  - *The Liu and Layland* condition (*LL*) introduced in [26].
  - *Increasing Period* condition (*IP*) [12].
  - *Utilization Oriented* condition (*UO*) developed by Y. Oh *et al.* [31].

- *Period-aware schedulability conditions*. Some variants of the utilization-based conditions use additional information from the task set in order to derive the utilization bound. In these conditions, the value of the *periods of the tasks* is included in the analysis. According to the way they derive their schedulability bounds, these conditions can be further classified as *closed-form period-aware* conditions and *non-closed-form period-aware* conditions:
  - *Closed-form period-aware conditions: Period Oriented* (*PO*) [7], Conditions based on *Harmonic Chains* [18], [17], [14] and *CRMB* [28].
  - *Non-closed-form period-aware conditions: T-Bound* and *R-Bound* [22], Algorithms of *Chen, Mok and Kuo* [8], *Sr* and *DCT* [14], and conditions that use *linear programming* techniques, such as the *PSUB* [32] and *LP* conditions [23].

## IV. Schedulability conditions for fixed-priority scheduling on a single processor

In this section, we review the best-known schedulability conditions found in the literature for Rate-Monotonic on one processor.

### A. Exact Schedulability Conditions for Rate-Monotonic

After Liu and Layland derived the RM scheduling algorithm along with its inexact condition, many *necessary* and *sufficient* tests for RM on one processor have been proposed [24][15][2][8][6][29][11]. In this section, we will review two of them.

*1) Exact Schedulability Condition Based on Processor's Demand (LE):* One of the first exact conditions was proposed by Lehoczky *et al.* [24]. In this test, the total demand of the processor time by a job in a critical instant is computed along with the total demand of the processor time for all the higher priority tasks. Then, the test checks if this demand can be met before the deadline of the job. The *LE* scheduling condition is formally defined in Theorem 1 [24].

**Theorem 1.** *(LE Condition) Let $\tau$= {$\tau_1$,$\tau_2$,..., $\tau_n$} be a task set with n tasks and $T_1 \leq T_2 \leq ... \leq T_n$. $\tau_i$ can be schedulable under RM if and only if,*

$$L_i = min_{\{t \in S_i\}} \left( \frac{W_i(t)}{t} \right) \leq 1 \qquad (1)$$

*where*

| $\tau_i$ | $\tau_1$ | $\tau_2$ | $\tau_3$ | $\tau_4$ | $\tau_5$ |
|---|---|---|---|---|---|
| $T_i$ | 8 | 16 | 3 | 12 | 48 |
| $C_i$ | 1 | 3 | 1 | 2 | 6 |
| $u_i$ | 0.125 | 0.1875 | 0.333 | 0.1666 | 0.125 |
| $U$ | 0.125 | 0.3125 | 0.6458 | 0.8124 | 0.9374 |

Table I
EXAMPLE TASK SET

$$S_i = \left\{ kT_j \mid j = 1, \ldots, i;\ k = 1, \ldots, \left\lfloor \frac{T_i}{T_j} \right\rfloor \right\},$$

$$W_i(t) = \sum_{j=1}^{i} C_j \left\lceil \frac{t}{T_j} \right\rceil$$

*The entire task set can be schedulable under RM if and only if*

$$L = max_{\{1 \le i \le n\}}\ L_i \le 1 \tag{2}$$

It can be observed that the computational complexity of the *LE* condition is *pseudo-polynominal* [24].

The function $L_i(t)$ is monotonically decreasing since $\frac{\left\lceil \frac{t}{T_i} \right\rceil}{t}$ is strictly decreasing except at a finite set of values called *rate-monotonic scheduling points*. When $t$ is a multiple of one of the periods $T_j$, for $1 \le j \le i$, the function has a local minimum [24]. Consequently, only a search over these local minimum values (the multiples of $T_j \le T_i$, $1 \le j \le i$) is needed, to determine if $\tau_i$ can meet its deadline.

**Example 1.** Table I shows a task set $\tau$ with five tasks, including its timing constraints: $C_i$, $T_i$, $u_i$ and $U = \sum_{\{j=1,..,i\}} u_i$. In order to verify if this task set is schedulable under the RM algorithm, we will use the exact *LE* condition.

We first sort the task set by the ascending period values. Thus, $\tau_1'$= *(3, 1)*, $\tau_2'$ = *(8, 1)*, $\tau_3'$ = *(12, 2)*, $\tau_4'$ = *(16, 3)* and $\tau_5'$ = *(48, 6)*. To determine if the task set is schedulable we just need to check if $\tau_5'$ fulfills its timing constraint. The set of scheduling points is $S_5$ = *{3, 6, 8, 9, 12, 15, 16, 18, 21, 24, 27, 30, 32, 33, 36, 39, 40, 42, 45, 48}*.

Task $\tau_5'$ is schedulable if any of the following equations hold (*for* $D_i = T_i$):

if $W_5(3) = C_1 + C_2 + C_3 + C_4 + C_5 \le T_1$ $\quad 13 > 3$
or $W_5(6) = 2C_1 + C_2 + C_3 + C_4 + C_5 \le 2T_1$ $\quad 14 > 6$
or $W_5(8) = 3C_1 + C_2 + C_3 + C_4 + C_5 \le T_2$ $\quad 15 > 8$
...
or $W_5(45) = 15C_1 + 6C_2 + 4C_3 + 3C_4 + C_5 \le 15T_1$ $\quad 44 \le 45$
or $W_5(48) = 16C_1 + 6C_2 + 4C_3 + 3C_4 + C_5 \le T_5$ $\quad 46 \le 48$

From the previous analysis, note that $W_5(t) \le t \le T_5$ (*t=45 and t=48)*. Therefore, we can conclude that the task set shown in Table I is schedulable under the RM algorithm.

*2) Exact Schedulability Condition Based on the Task's Response Times:* Joseph and Pandya introduced an exact schedulability condition in [15] for fixed priority scheduling. In this test, the response time of each task $r_i$ is obtained, and if $r_i \le D_i$, then task $\tau_i$ meets its deadline.

This test starts by obtaining the response time of the highest priority task, using the following equation:

$$r_i = C_i + \sum_{j\ \epsilon\ hp(i)} \left\lceil \frac{r_i}{T_j} \right\rceil C_j$$

where *hp(i)* is the set of tasks with a higher priority than the task $\tau_i$. Since $r_i$ appears on both sides of the equation, a possible solution was proposed by Audsley *et al.* in [2]. The solution is obtained by the following iterative process:

$$r_i^{n+1} = C_i + \sum_{j\ \epsilon\ hp(i)} \left\lceil \frac{r_i^n}{T_j} \right\rceil C_j \tag{3}$$

Iterations described in Eqn. 3 can start considering $r_i^0 = \sum_{k=1}^{i} C_k$. It is easy to note that $r_i^{n+1} \ge r_i^n$. If $r_i^n \ge D_i$, then task $\tau_i$ will miss its deadline. However, if $r_i^{n+1} = r_i^n$, the iterative process will conclude, meaning that $\tau_i$ is schedulable.

The response time analysis has evolved to include offsets, blocking, fault tolerance, and release jitter [2].

The exact schedulability analysis is time-consuming due to its high computational complexity. Therefore, it is not suitable for online schedulability analysis.

### B. Liu and Layland (LL) Schedulability Condition

In [26], Liu and Layland defined the *critical instant* for a task as the instant at which a request for that task will have the largest response time, and showed that if all the tasks meet their deadlines at their critical instants, then the task set is feasible. The worst-case phasing occurs when $\theta_1 = \theta_2 = \cdots = \theta_n$ (e.g. $\theta_i = 0$ for all $i$).

Liu and Layland introduced the concept of *utilization factor* in [26] and defined it as the fraction of the processor time spent in the execution of the task set. Further, they defined that a task set is said to *fully utilize* the processor according to a given scheduling algorithm if the set of tasks can be feasibly scheduled and that any increase in the execution time of any of the tasks will make the task set infeasible with respect to that algorithm. For a given fixed-priority scheduling algorithm, the *least upper bound* of the utilization factor is the minimum of the *utilization factors* over all the sets of tasks that *fully utilize* the processor.

In order to derive the *least upper bound* for the Rate Monotonic algorithm, Liu and Layland showed that the worst-case situation occurs when the task set fully utilizes the processor, all tasks start simultaneously (that is, at its critical instant) and the relationship among the periods is such that $\forall_i = 2, \ldots, n\ T_1 < T_i < 2T_1$. Under this worst-case scenario, Liu and Layland found the *least upper bound* by minimizing the total utilization with respect to the period values.

The *Liu and Layland Condition (LL)* is formalized in Theorem 2 [26].

**Theorem 2.** *(LL Condition) A set of tasks $\tau$ is schedulable under the RM algorithm if the following condition is satisfied*

$$U \le n(2^{\frac{1}{n}} - 1) \tag{4}$$

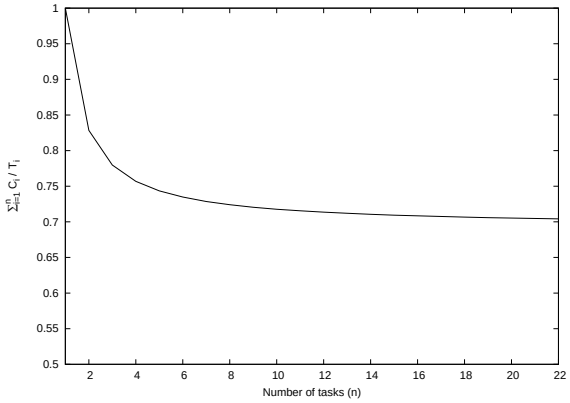


Figure 1. Performance of the *LL* condition

| $\tau_i$ | $T_i$ | $C_i$ | $u_i$ | $U_i$ | *LL* bound |
|---|---|---|---|---|---|
| $\tau_1$ | 8 | 1 | 0.125 | 0.1250 | *1* |
| $\tau_2$ | 16 | 3 | 0.1875 | 0.3125 | *0.8284* |
| $\tau_3$ | 3 | 1 | 0.3333 | 0.6458 | *0.7798* |
| $\tau_4$ | 12 | 2 | 0.1666 | 0.8124 | *0.7568* |
| $\tau_5$ | 48 | 6 | 0.1250 | 0.9374 | *0.7435* |

Table II
*LL* CONDITION APPLIED TO THE TASK SET OF TABLE I

If the condition of Theorem 2 is not satisfied, that is, $U > n(2^{\frac{1}{n}} - 1)$, then it is not known whether the task set is schedulable under Rate-Monotonic.

It is important to note that the *LL* condition depends only on the *number of tasks* in the system [26]. The computational complexity of the *LL* condition is *O(n)*.

Fig. 1 shows the processor utilization factor under the *LL* schedulability condition. It can be observed that when the number of tasks tends to infinity, the *minimum achievable utilization* factor tends to *ln(2) = 0.6931*.

Leung and Whitehead, in [25], generalized the results provided by Liu and Layland and proved that the *Deadline Monotonic (DM) algorithm* is optimal for the fixed-priority scheduling model. In the DM scheduling algorithm, task deadlines can be smaller than its periods ($D_i \leq T_i$).

**Example 2.** After applying the *LL* condition to the task set shown in Table I, we can conclude that tasks $\tau_1$, $\tau_2$, and $\tau_3$ can be feasibly scheduled, but adding $\tau_4$ and $\tau_5$ violates the *LL* condition, as shown in Table II.

### C. Increasing Period (IP) Schedulability Condition

The *IP* condition was introduced by Dhall and Liu [12] and was proposed to be used together with the multiprocessor algorithms *Rate-Monotonic Next Fit* and *Rate-Monotonic First-Fit*. In order to determine the tasks that can be assigned to each processor, the *IP* condition takes into account both the utilization of the task set assigned to a processor and the utilization of the new task. The *IP* schedulability condition is defined in Theorem 3 [12]:

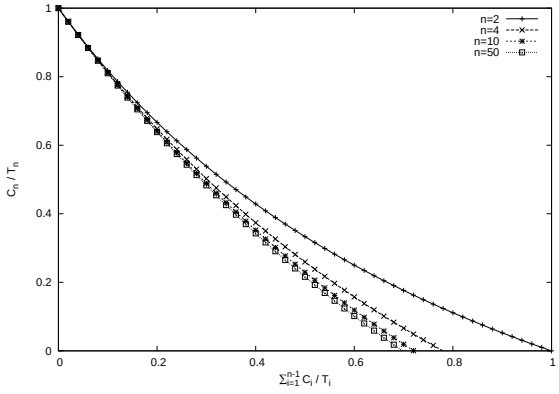


Figure 2. Performance of the *IP* Condition

**Theorem 3.** *(IP Condition) Let $\tau$ = {$\tau_1$, $\tau_2$, ..., $\tau_n$} be a set of n tasks with $T_1 \leq T_2$ ... $\leq T_n$ and let*

$$U_{n-1} = \sum_{i=1}^{n-1} \frac{C_i}{T_i} \leq (n-1)(2^{\frac{1}{(n-1)}} - 1) \tag{5}$$

*If the following condition is met*

$$u_n \leq 2\left(1 + \frac{U_{n-1}}{(n-1)}\right)^{-(n-1)} - 1 \tag{6}$$

*then the set of tasks can be feasibly scheduled under the RM algorithm. When $n \rightarrow \infty$, the minimum utilization of task $\tau_n$ approaches to (2 $e^{-u}$ - 1).*

This condition requires an ordering of the periods of the tasks. Because of this ordering, its complexity is *O(n log n)*. As can be noticed, this condition is based on the *utilization* and the *number of tasks* in the system.

Fig. 2 shows the performance of the *IP* condition, where the utilization of task $\tau_n$ is a function of the *(n-1)* tasks already in the system. The different curves illustrate different values for the number of tasks (*n*). The area under the curve denotes the feasibility area for this test.

**Example 3.** In this example, we will show the performance of the *IP* schedulability condition using the task set described in Table I. To use this condition, tasks must be sorted in the non-decreasing order of their periods. After applying the *IP* condition, we note, in Table III, that while tasks $\tau_3$, $\tau_1$, $\tau_4$ , and $\tau_5$ are identified as schedulable, the *IP* condition fails to identify task $\tau_2$ as schedulable.

| $\tau_i$ | $T_i$ | $C_i$ | $u_i$ | *IP* bound |
|---|---|---|---|---|
| $\tau_3$ | 3 | 1 | 0.3333 | - |
| $\tau_1$ | 8 | 1 | 0.1250 | 0.5000 |
| $\tau_4$ | 12 | 2 | 0.1666 | 0.3238 |
| $\tau_2$ | 16 | 3 | 0.1875 | 0.1336 |
| $\tau_5$ | 48 | 6 | 0.1250 | 0.1336 |

Table III
*IP* CONDITION APPLIED TO THE TASK SET OF TABLE I

### D. Period Oriented (PO) Schedulability Condition

Burchard *et al.* [7] introduced the *Period Oriented* condition to be used by the *Rate Monotonic Small Tasks* (*RMST*) and *Rate Monotonic General Tasks* (*RMGT*) multiprocessor algorithms. To be able to use the *PO* condition, it is necessary to know the values of the periods of the tasks in the system. The *PO* condition is formally defined in Theorem 4 [7]:

**Theorem 4.** *(PO Condition) Given a set of tasks* $\tau$= {$\tau_1$, $\tau_2$, ..., $\tau_n$}, $S_i$ *and* $\beta$ *are defined as follows:*

$$S_i = \log_2 T_i - \lfloor \log_2 T_i \rfloor \qquad i = 1, \ldots, n \tag{7}$$

*and*

$$\beta = \max_{1 \le i \le n} S_i - \min_{1 \le i \le n} S_i \tag{8}$$

*(a) if* $\beta < (1 - \frac{1}{n})$ *and the total utilization satisfies that*

$$U \le (n-1)\,(2^{\beta/(n-1)} - 1) + 2^{1-\beta} - 1 \tag{9}$$

*then the task set is schedulable on one processor under RM.*

*(b) if* $\beta \ge (1 - \frac{1}{n})$ *and the total utilization satisfies that*

$$U \le n(2^{1/n} - 1) \tag{10}$$

*then the task set is schedulable on one processor under RM.*

From Eqn. 7 it can be observed that $S_i$ is a function that goes from zero (when the period $T_i$ is a power of two) to one (when the period $T_i$ is the next power of two), and it measures the logarithmic distance of the period of task $\tau_i$ from a power of two (where 0.5 means that the period is logarithmically in the middle of two powers of two). Therefore, $\beta$ measures how logarithmically equidistant the periods of all tasks are from a power of two. When $\beta$ = 0, $T_{i+1}$= $2\alpha T_i$ $\forall$ *i=1,2,..., n*; and $\forall$ $\alpha \in \mathbb{N}$, and $\alpha$ > 0.

As $\beta$ approaches to zero, the utilization bound tends to one, independent of the number of task. On the other hand, as $\beta$ approaches to one, the utilization bound approaches the *LL* condition.

A simpler version of Eqn. 9 of the *PO* condition is defined in Corollary 1 [7].

**Corollary 1.** *(PO Condition) Given a set of tasks* $\tau$= {$\tau_1$, $\tau_2$, ..., $\tau_n$} *and given* $\beta$ *(as defined in Theorem 4), if the total utilization satisfies that*

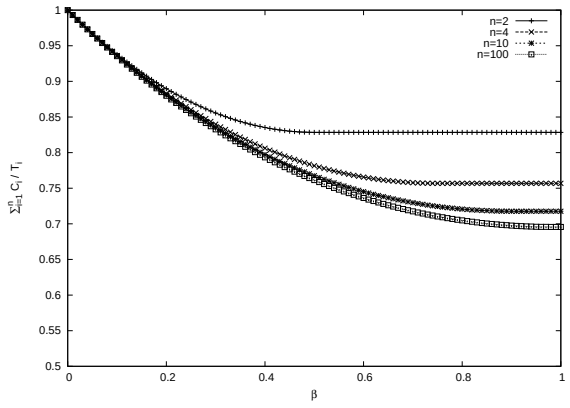


Figure 3. Performance of the *PO* Condition

$$U \le \max\{\ln 2, 1 - \beta \ln 2\} \tag{11}$$

*then the task set can be feasibly scheduled on one processor under RM.*

Fig. 3 shows the performance of the *PO* condition. Each curve shows, for a given number of tasks, the relationship between $\beta$ and the utilization of the task set. The area under the curve denotes the feasibility area for this test. It can be observed that when the number of tasks is large and the value of $\beta$ = 1, then the minimum achievable utilization is approximately 69%, similar to the result provided by the *LL* condition.

**Example 4.** The first step on applying the *PO* condition to the task set described in Table I involves calculating the $S_i$ values using Eqn. 7, and sorting them in the non-decreasing order. The obtained values are $S_i$ = {$S_1$= *0,* $S_2$= *0,* $S_3$= *0.5849,* $S_4$= *0.5849,* $S_5$= *0.5849*}, and $\beta$ = *0.5849* (from Eqn. 8). Because $\beta < (1 - \frac{1}{n})$ *(0.5848 < 0.8)*, Corollary 1 can be used to check the schedulability of the task set. Since 0.9375 > 0.6931, the *PO* condition fails to identify the task set as schedulable.

### E. Utilization Oriented (UO) Schedulability Condition

Y. Oh *et al.* [31] introduced a schedulability condition based on the values of tasks utilization $u_i$. Oh *et al.* derived their schedulability condition from the worst-case scenario identified by Liu and Layland [26], but instead of minimizing the total utilization with respect to the period values, they derived their schedulability condition as a function of the individual task utilization [31].

The *UO* condition was proposed to be used in the *Rate-Monotonic-First-Fit-Decreasing-Utilization (RM-FFDU)* multiprocessor algorithm and is defined in Theorem 5 [31].

**Theorem 5.** *(UO Condition) Let* $\tau$ = {$\tau_1$,$\tau_2$,..., $\tau_{n-1}$} *be a task set of (n-1) tasks, feasibly scheduled under RM. A new task* $\tau_n$ *can be feasibly scheduled along with the (n-1) tasks already in the system (on one processor under RM), if the following condition is met:*

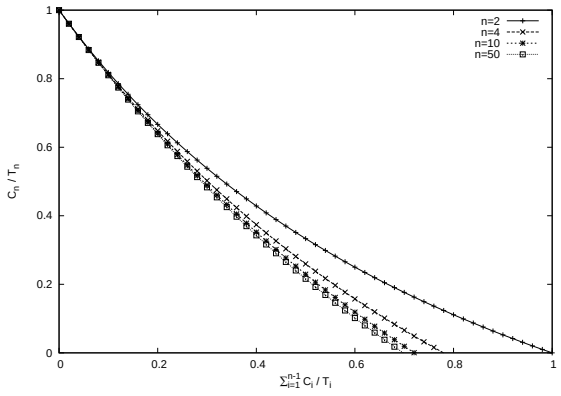


Figure 4. Performance of the *UO* Condition

$$\frac{C_n}{T_n} \leq 2 \left[ \prod_{i=1}^{n-1} (1 + u_i) \right]^{-1} - 1 \tag{12}$$

Fig. 4 shows the *UO* utilization bound for different values of *n*, where the *x-axis* denotes the utilization of the *(n-1)* tasks already in the system, and the *y-axis* denotes the utilization of task $\tau_i$. The area under each curve denotes the feasibility area for this test. Note that this condition takes into account the number of tasks and the individual utilization of the tasks. The complexity of this condition is *O(n)*.

Bini *et al.* [5] introduced the *Hyperbolic Bound (HB)* condition, a schedulability test similar to the one provided by the *UO* condition. The *HB* condition is expressed in Theorem 6 [5].

**Theorem 6.** *(HB Condition) Let $\tau$ = {$\tau_1$,$\tau_2$,..., $\tau_n$} be a set of n periodic tasks, where each tasks $\tau_i$ is characterized by a processor utilization $u_i$. Then, $\tau$ is schedulable by the RM algorithm if*

$$\prod_{i=1}^{n} (u_i + 1) \leq 2 \tag{13}$$

It is clear that Eqn. 13 can be derived from Eqn. 12. Bini *et al.* extended the *HB* condition to include resource sharing and aperiodic servers.

**Example 5.** After using the *UO* condition to verify the schedulability of the task set described in Table I, we find that tasks $\tau_1$, $\tau_2$, and $\tau_3$ are proved to be schedulable, but the *UO* condition fails to identify tasks $\tau_4$ and $\tau_5$ as schedulable, as shown in Table IV.

### *F. T-Bound and R-Bound Schedulability Conditions*

Lauzac *et al.* [21] developed the *T-Bound* and *R-Bound* schedulability conditions to be used as an admission control for RM scheduling on uniprocessor systems, and extended their results to be used as an admission control for the multiprocessor systems. While discussing the *LL* schedulability condition, we noted that the worst-case scenario occurs when all the tasks start simultaneously and the relationship among the periods is such that the ratio between any task periods is less than two. Liu and Layland showed in [26] that under this scenario, the computation times used to derive the *least upper bound* are:

$$C_i = T_{i+1} - T_i \ (i = 1, \ldots, n-1) \ and \ C_n = 2T_1 - T_n$$

If the total utilization $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$ is rewritten using these computation times, a new schedulability bound for RM can be derived. This bound is shown in Lemma 1 [21].

**Lemma 1.** *Given a task set $\tau$ of m tasks ordered by increasing periods, and the restriction that the ratio between any task periods is less than 2, $\tau$ is schedulable if*

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \ \leq \ \sum_{i=1}^{n-1} \left[ \frac{T_{i+1}}{T_i} \right] + 2\frac{T_1}{T_n} - n \tag{14}$$

The *T-Bound* condition uses the *ScaleTaskSet* algorithm to transform the original task set into an equivalent task set where the ratio between the maximum and minimum periods is less than 2 (that is, *r = $T_{max}$ / $T_{min}$ < 2*). Using this transformed task set, the condition verifies its schedulability through Eqn. 14. As stated in Lemma 2 [21], if the transformed task set is feasibly scheduled under RM then the original task set is also feasible.

**Lemma 2.** *Let $\tau$ be a given periodic task set, and let $\tau'$ be the transformed task set after applying the ScaleTaskSet algorithm to $\tau$. If $\tau'$ is schedulable on one processor under RM, then $\tau$ is also schedulable.*

The *ScaleTaskSet* algorithm is defined in Algorithm 1 and the *T-Bound* condition is formally defined in Theorem 7 [21].

| $\tau_i$ | $T_i$ | $C_i$ | $u_i$ | *UO* bound |
|---|---|---|---|---|
| $\tau_3$ | 3 | 1 | 0.3333 | - |
| $\tau_1$ | 8 | 1 | 0.1250 | 0.5000 |
| $\tau_4$ | 12 | 2 | 0.1666 | 0.3333 |
| $\tau_2$ | 16 | 3 | 0.1875 | 0.1429 |
| $\tau_5$ | 48 | 6 | 0.1250 | 0.1429 |

Table IV
*UO* CONDITION APPLIED TO THE TASK SET OF TABLE I

**Algorithm 1** *ScaleTaskSet* Algorithm

ScaleTaskSet (**In**: $\tau$, **Out**: $\tau'$)
**begin**
  Sort the task set in $\tau$ by increasing period;
  **for** (*i*=1 **to** *n* - 1) **do**
    $T'_i = T_i \, 2^{\left\lfloor \log \frac{T_n}{T_i} \right\rfloor}$;
    $C'_i = C_i \, 2^{\left\lfloor \log \frac{T_n}{T_i} \right\rfloor}$;
  Sort the task set in $\tau'$ by increasing period;
  **return** ($\tau'$);
**end**

| $\tau_i$ | $T_i$ | $C_i$ | $T'_i$ | $C'_i$ | $U'_i$ | *T-Bound* |
|---|---|---|---|---|---|---|
| $\tau_3$ | 3 | 1 | 32 | 4 | 0.1250 | - |
| $\tau_1$ | 8 | 1 | 32 | 6 | 0.3125 | 1 |
| $\tau_4$ | 12 | 2 | 48 | 16 | 0.6458 | 0.8333 |
| $\tau_2$ | 16 | 3 | 48 | 8 | 0.8124 | 0.8333 |
| $\tau_5$ | 48 | 6 | 48 | 6 | 0.9374 | 0.8333 |

Table V
*T-Bound* CONDITION APPLIED TO THE TASK SET OF TABLE I

**Theorem 7.** *(T-Bound Condition) Consider a periodic task set $\tau$, and let $\tau'$ be the transformed task set after executing the ScaleTaskSet algorithm to $\tau$. If Eqn. 14 holds for $\tau'$, then the task set $\tau$ can be feasibly scheduled on one processor under RM.*

It is important to note that the *T-Bound* condition uses the value of the periods $T'_1,\ldots,T'_n$, derived by the *ScaleTaskSet* algorithm. However, in order to provide an admission control criterion that does not depend on the periods of all the tasks, Lauzac *et al.* [21], [22] derived the *R-Bound* schedulability condition, which uses the relationship between the largest and the smallest period values in the task set. The *R-Bound* schedulability condition is defined in Theorem 8 [22].

**Theorem 8.** *(R-Bound Condition) Consider a periodic task set $\tau$, and let $\tau'$ be the transformed task set after applying the ScaleTaskSet algorithm to $\tau$. If,*

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq (n-1)(r^{\frac{1}{(n-1)}} - 1) + \left(\frac{2}{r}\right) - 1 \tag{15}$$

where $r = \frac{T'_n}{T'_1}$, *the task set $\tau$ can be feasibly scheduled on one processor under RM.*

Because the *T-Bound* condition uses more information about the task set, it outperforms the *R-Bound* condition. However, Lauzac *et al.* showed in [22] that when *r* is close to one, the performance of the *R-Bound* condition is similar to the performance of the *T-Bound* condition. The complexity of the *T-Bound* and *R-Bound* conditions is *O(n log n)*.

**Example 6.** Before applying the *T-Bound* condition to the task set described in Table I, we first need to generate a transformed task set using the *ScaleTaskSe*t algorithm. Then, according to Theorem 7, if the transformed task set is schedulable under RM, the former task set is also schedulable. Table V shows the values of the periods and the execution times of the transformed tasks. It can be noted that under the *T-Bound* condition tasks $\tau_1$, $\tau_2$, $\tau_3$, and $\tau_4$ are proved to be schedulable, whereas the *T-Bound* condition fails to identify task $\tau_5$ as schedulable.

### *G. Harmonic Chains (HC) Schedulability Condition*

Kuo and Mok [18] extended the results provided by Liu and Layland [26], by relating the *achievable utilization factor* to the number of *harmonic chains* found in a task set. A harmonic chain is a list of numbers (periods) wherein each number divides every number after it [8].

Kuo and Mok [18] developed the *Harmonic Chain* (*HC*) condition, in which a periodic task set $\tau$ will find a feasible schedule if its utilization factor is no larger than $k(2^{\frac{1}{k}} - 1)$, where *k* is the size of the harmonic base of $\tau$.

The harmonic chains found in the task set conform the *harmonic base* of a task set. The definition of *harmonic base* is described as follows:

**Definition 1.** *(Harmonic Base of $\tau$) Let S be the set of periods (positive numbers) of a set of periodic tasks $\tau$. A subset H of S is said to be a harmonic base of the task set $\tau$ if there is a partition, say $\Gamma$, of S into |H| subsets such that:*

*1. Each member of H is the smallest element in exactly one member of the partition $\Gamma$, and*

*2. If x and y are two elements in the same member of the partition $\Gamma$, then either x divides y or y divides x.*

*Each subset in the partition $\Gamma$ is called a harmonic chain [18].*

In order to explain the *HC* condition, we will use the following example:

Let *T* be a task set where every task is defined as $\tau_i = (T_i, C_i)$. We have *T* = { $\tau_1$ = (3, 1), $\tau_2$ = (5, 1), $\tau_3$ = (15, 2), $\tau_4$ = (20, 3), $\tau_5$ = (60, 8) }. Let *P* be the set of periods from *T*, such that *P* = {3,5,15,20,60}. The subset *H* = {3,5} is a *harmonic base* of *P* because there exists a partition $\Gamma$ in |H| subsets, namely $\Gamma$ = {{3,15}, {5,20,60}}, such that:

(1) each member of *H* is the smallest single element of the partition $\Gamma$, and
(2) for each par of elements within the partition $\Gamma$, one element divides the other.

The *Harmonic Chains* condition is formally defined in Theorem 9 [18].

**Theorem 9.** *(HC Condition) Let $\tau$ be a set of periodic tasks and let k be the size of the harmonic base of $\tau$. If the utilization factor is no larger than $k\,(2^{\frac{1}{k}} - 1)$, then $\tau$ is schedulable by a preemptive fixed priority scheduler.*

A polynomial time algorithm can solve the problem of computing the harmonic base of a periodic task set. From Theorem 9, it can be observed that when the size of the harmonic base is small, the utilization bound is large. For instance, for *k=1*, though the utilization may be as high as 100%, the task set is guaranteed to be schedulable. Note thatthe *HC* condition is similar to the *LL* condition when the periods of all tasks are *relative primes*[1].

**Example 7.** The task set described in Table I has two *harmonic chains*: $\Gamma = \{\{8, 16\}, \{3, 12, 48\}\}$, as shown in Fig. 5. After applying the *HC* condition, we concluded that this condition identifies tasks $\tau_1$, $\tau_2$, $\tau_3$, and $\tau_4$ as schedulable, since their total utilization is not higher than $2\,(2^{\frac{1}{2}} - 1)$. Task $\tau_5$ violates the *HC* condition since $\sum_{i=1}^{5} u_i > 2\,(2^{\frac{1}{2}} - 1)$, therefore, it is not identified as schedulable.

[1] *Two numbers a and b are relative primes if they are non zeros and MCD(a, b) = 1*

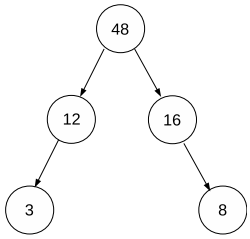


Figure 5. The *harmonic chains* in the task set described in Table I

### H. Root Condition

Kuo *et al.* [17] developed the *Root* condition and demonstrated that a task set can be feasibly scheduled as long as the utilization of the task set is no larger than $R(2^{\frac{1}{R}}-1)$, where *R* is the number of *roots* in the task set. The concept of *root* is defined next [18].

**Definition 2.** *Let $\tau$ = {$\tau_1$,$\tau_2$,..., $\tau_n$} be a periodic task set. Task $\tau_i$ is a root in $\tau$ if there does not exist any task period in $\tau$, which is larger than and divisible by the period of the task $\tau_i$.*

In order to explain the concept of *root*, we will use the example described in the previous subsection:

Let $\tau$ be a task set where every task is defined as $\tau_i = (T_i, C_i)$. We have $\tau$ = { $\tau_1$ = (3, 1), $\tau_2$ = (5, 1), $\tau_3$ = (15, 2), $\tau_4$ = (20, 3), $\tau_5$ = (60, 8) }. Let *P* be the set of periods from $\tau$, such that, *P* = {3,5,15,20,60}. The *harmonic base* of *P* is $\Gamma$ = {{3,15}, {5,20,60}}. From the harmonic base of *P* we can observe that 60 is a value such that there does not exist any task period in *P*, which is larger than and divisible by 60.

The *Root* condition is defined in Theorem 10 [17].

**Theorem 10.** *(Root Condition) Suppose that the task set {$\tau_1$,$\tau_2$,..., $\tau_{i-1}$} is schedulable. Let R be the number of roots in the task set $\tau$ = {$\tau_1$,$\tau_2$,..., $\tau_{i-1}$, $\tau_i$}. If the total utilization factor of $\tau$ is no larger than $R(2^{\frac{1}{R}}-1)$, then $\tau$ is schedulable.*

Because the number of roots could be much less than the number of tasks, and the size of its harmonic base, it is expected that the *Root* condition improves the acceptance ratio of the *LL* and *HC* conditions. This can be observed in Corollaries 2, 3 and 4 [17].

**Corollary 2.** *Let $\tau$ be a set of periodic tasks. If $\tau$ is guaranteed to be scheduled according to the LL condition, then $\tau$ is guaranteed to be scheduled according to the Root condition.*

**Corollary 3.** *Let $\tau$ be a set of periodic tasks. If $\tau$ is guaranteed to be scheduled according to the HC condition, then $\tau$ is guaranteed to be scheduled according to the Root condition.*

**Corollary 4.** *There exists a task set that is guaranteed to be scheduled according to Root condition, but not according to the LL and HC conditions.*

An important feature of the *Root* condition is that it was developed to be used incrementally for *online* admission control.

**Algorithm 2** *Sr* Algorithm

**Input:** $\tau = \{\tau_i = (C_i, T_i) \mid 1 \le i \le n\}$, where $\tau$ is a periodic task set and $T_i \le T_j$, $\forall\, i < j$;
**Output:** Task set $\tau'$ and $\Phi_\tau(r)$;
**begin**
  **for** (*i*=1 **to** *n*) **do** $l_i = \frac{T_i}{2^{\lceil log(T_i/T_1)\rceil}}$;
  sort ($l_1$,$l_2$,...,$l_n$) into non-decreasing order and remove duplicates, let ($k_1$,$k_2$,...,$k_u$) be the resulting sequence;
  **for** (*i*=1 **to** *n*) **do** put $\tau_i$ into subset $\pi_{l_i}$;
  **for** (*j*=1 **to** *u*) **do** $U(\pi_{k_j}) = \sum_{\tau_i \in \pi_{k_j}} \frac{C_i}{T_i}$;
  compute $\Phi_\tau(k_u) = \Phi_\tau(T_1)$;
  **for** (*j*=*u*-1 **down-to** 1) **do**
    $\Phi_\tau(k_j) = \frac{k_{j+1}}{k_j}\Phi_\tau(k_{j+1}) - U(\pi_{k_j})$;
  find $r^*$ such that $\Phi_\tau(r^*) = \min_{r \in \{k_1,k_2,\ldots,k_u\}} \Phi_\tau(r)$;
  **for** (*i*=1 **to** *n*) **do** $T'_i = r^* \cdot 2^{\lfloor log(T_i/r^*)\rfloor}$;
  **return** $\Phi_\tau(r^*)$ and $\tau'$;
**end**

**Example 8.** In this example, we will show the performance of the *Root* condition. As described in the previous example, the task set from Table I has two *harmonic chains*: $\Gamma$ = {{8,16}, {3,12,48}}. However, it has only one *root*, *R*=48, as shown in Fig. 5. Therefore, after applying the *Root* condition, we observe that tasks $\tau_1$, $\tau_2$, $\tau_3$, $\tau_4$, and $\tau_5$ are identified as schedulable, since their total utilization is no larger than 1 ( $1(2^{\frac{1}{1}}-1)$ ).

### I. Sr and DCT Schedulability Conditions

Han and Tyan [14] introduced two polynomial-time schedulability tests that transform the task periods into a special pattern where all the periods belong to a single harmonic chain. According to Theorem 9, when *k*=1 (which means there is only one harmonic chain in $\tau$), the transformed task set $\tau'$ is schedulable if its total utilization is less than or equal to 1. Han and Tyan proved that if $\tau'$ is schedulable under RM, then the original task set $\tau$ is also schedulable under RM. The transformed task set $\tau'$ must satisfy the Condition 1 [14].

*Condition 1. $\tau'_i \le \tau_i$, for all i = 1, 2,..., n, and $\tau'_i$ evenly divides $\tau'_{i+1}$, denoted as $\tau'_i | \tau'_j$, (thus, $\tau'_i \le \tau'_{i+1}$) for all i = 1, 2,..., n.*

The schedulability of the transformed task set $\tau'$ is defined in Theorem 11 [14].

**Theorem 11.** *Given a task set $\tau$, if there exists another task set that satisfies Condition 1 and $U_{\tau'} = \sum_{i=1}^{n} \frac{C_i}{T_i} \le 1$, then $\tau$ is schedulable by RM.*

In order to apply the results provided by Theorem 11, the problem is, given a task set $\tau$, how to find (in polynomial time) another task set that satisfies Condition 1 and whose utilization $U_{\tau'}$ is as small as possible. Han and Tyan proposed, in [14], the *Sr* and *DCT* algorithms to find such $\tau'$.

| $\tau_i$ | $T_i$ | $C_i$ | $T'_i$ | $C'_i$ | $u'_i$ | $U'_i$ |
|---|---|---|---|---|---|---|
| $\tau_3$ | 3 | 1 | 3 | 1 | 0.3333 | 0.3333 |
| $\tau_1$ | 8 | 1 | 6 | 1 | 0.1666 | 0.5000 |
| $\tau_4$ | 12 | 2 | 12 | 2 | 0.1666 | 0.6667 |
| $\tau_2$ | 16 | 3 | 12 | 3 | 0.2500 | 0.9167 |
| $\tau_5$ | 48 | 6 | 48 | 6 | 0.1250 | 1.0417 |

Table VI
*Sr* AND *DCT* CONDITIONS APPLIED TO THE TASK SET OF TABLE I

*1) Sr Algorithm:* The first algorithm proposed is called the *Sr algorithm (Specialization Operation*). In this algorithm, each period $T_i$ of the task set $\tau$ is transformed into another period $T'_i = r \cdot 2^{\lfloor log(T_i/r) \rfloor}$, where *r* is a real number chosen from the range $(\frac{T_1}{2}, T_1]$. Because $T'_i \leq T_i$ for all *i* and $T'_i | T'_j$ for all $i < j$, the transformed period values belong to a single harmonic chain. Furthermore, because $T'_i \leq T_i$ for all *i*, we have that $U_{\tau'} = \sum_{i=1}^{n} \frac{C'_i}{T'_i} \geq U_\tau = \sum_{i=1}^{n} \frac{C_i}{T_i}$. To minimize the total utilization increase $\triangle = U_{\tau'} - U_\tau$, the value of *r* should be carefully chosen. The *Sr* algorithm, reproduced in Algorithm 2, finds the best value for *r*, and then derives the new periods $T'_i$, for all *i*, using the best *r*.

The *Sr algorithm* first computes $l_i = T_i / 2^{\lceil log(T_i/T_1) \rceil}$, for 1 ≤i ≤n, where $\frac{T_1}{2} < l_i \leq T_1$, naming $k_1 < k_2 < \ldots < k_u$, $u \leq n$, the sorted sequence of $l'_i s$ with duplicates removed. Because $l_i = T_i$, we know that $k_u = T_1$. The sequence $\{k_1, k_2, \ldots, k_u\}$ is called the *special base* of $\tau$. The value of *r* that minimizes the total utilization increase $\triangle$, denoted by $r^*$, can always be found in the *special base*. The total utilization of the task set $\tau'$ with its periods $\{T'_1, T'_2, \ldots, T'_n\}$ specialized from $\{T_1, T_2, \ldots, T_n\}$ with respect to *r*, is called $\Phi_\tau(r)$, and $\Phi^*_\tau = \Phi_\tau(r^*) = \min_{\{\frac{T_1}{2} < r \leq T_1\}} \Phi_\tau(r)$. The algorithm computes $\Phi_\tau(k_j)$ for all $k_j$ in the *special base* of $\tau$, and then selects the one that results in the minimum value of $\Phi_\tau(k_j)$, and uses that $k_j$ as the value of *r* in the *specialization operation*. This *specialization operation* provides the periods for the transformed task set that belongs to a single fundamental frequency. Then, the utilization of the transformed task set is computed and if it is less than or equal to 1, the task set is schedulable. The complexity of the *Sr* condition is *O(n log n)*.

**Example 9.** In this example, we will show the performance of the *Sr* schedulability condition. The first step in the *Sr* algorithm is to find the $l_i$ values to obtain the special base of *r*. After computing the $l_i$ values and removing duplicates, we have $k_1 = 2$ and $k_2 = 3$. Next, we need to find the value of $r^*$ using $\Phi^*_\tau = \Phi_\tau(r^*) = \min_{\{\frac{T_1}{2} < r \leq T_1\}} \Phi_\tau(r)$ such that the total utilization increase is minimized. We observe that $\Phi_\tau(k_1) = 1.25$ and $\Phi_\tau(k_2) = 1.0417$, and therefore the value of $r^*$ to be used is $r^* = 3$. Once the $r^*$ value is found, we use it in the specialization operation to generate the transformed task set $\tau'$, which is shown in Table VI. It can be observed from Table VI that tasks $\tau_1$, $\tau_2$, $\tau_3$, and $\tau_4$ are identified as schedulable, since their total utilization is no larger than 1. However, task $\tau_5$ is not identified as schedulable by the *Sr* condition since $U'_5 > 1$.

**Algorithm 3** DCT Algorithm

**Input:** $\tau = \{\tau_i = (C_i, T_i) \mid 1 \leq i \leq n\}$, where $\tau$ is a periodic task set and $T_i \leq T_j$, $\forall\, i < j$;
**Output:** Task set $\tau'$;
**begin**
  min_f = -1; min_utilization=∞;
  **for** (*f*=1 **to** *n*) **do** {
    $Z_f = T_f$;
    **for** (*i*=*f*+1 **to** *n*) **do** $Z_i = Z_{i-1} \cdot \left\lfloor \frac{T_i}{Z_{i-1}} \right\rfloor$;
    **for** (*i*=*f*-1 **down-to** *1*) **do** $Z_i = \frac{Z_{i+1}}{\left\lceil \frac{Z_{i+1}}{T_i} \right\rceil}$;
    utilization = $\sum_{i=1}^{n} \frac{C_i}{Z_i}$;
    **if** utilization < min_utilization **then**
      min_utilization = utilization;
      min_f = f;
      **for** (*i*=1 **to** *n*) **do** $T'_i = Z_i$;
    **endif**
  }
**end**

*2) DCT Algorithm:* The second algorithm proposed by Han and Tyan in [14] is called the *DCT* algorithm. The idea behind the *DCT* algorithm is the following. For each *f*, $1 \leq f \leq n$, $T'_f = T_f$, and recursively, $T_i$, for each $i > f$, is transformed to the largest integral multiple of $T'_{i-1}$ that is less than or equal to $T_i$. That is,

$$T'_i = T'_{i-1} \cdot \left\lfloor \frac{T_i}{T'_{i-1}} \right\rfloor, \; for\, i = f+1, f+2, \ldots, n \qquad (16)$$

Similarly, $T_i$, for $i < f$, is recursively transformed to the largest divisor of $T'_{i+1}$ that is less than or equal to $T_i$. That is,

$$T'_i = \frac{T'_{i+1}}{\left\lceil \frac{T'_{i+1}}{T_i} \right\rceil}, \; for\, i = f-1, f-2, \ldots, 1 \qquad (17)$$

The value of $f$ that results in the minimum utilization increase will be the final index of $T_i$ whose transformed value of $T'_i$ will be fixed at $T_i$. The *DCT* algorithm is described in Algorithm 3. The complexity of the *DCT* condition is $O(n^2)$.

**Example 10.** In this example, we will show the performance of the *DCT* schedulability condition. After applying the *DCT* algorithm to the task set shown in Table I, we found that when *f=3*, *a minimum utilization increase = 1.0417* is obtained, which corresponds to the transformed task set $\tau'$ shown in Table VI. Using Theorem 11 to verify the feasibility of this task set, we conclude that tasks $\tau_1$, $\tau_2$, $\tau_3$, and $\tau_4$ are proved to be schedulable, since their total utilization is no larger than 1. However, task $\tau_5$ is not identified as schedulable by the *DCT* condition since $U'_5 > 1$. It can be observed that for the task set described in Table I, the *Sr* and *DCT* conditions produce identical results.

Han and Tyan showed in [14] that the *DCT* condition provides a better performance than the *Sr* condition. Han extended the *DCT* and *Sr* conditions to be used in the multiframe task model in [13].

**Algorithm 4** *Algorithm 1* of Chen, Mok, and Kuo

**Input**: **TaskPeriod**[n] in non-decreasing order;
**Output**: Utilization bound $\overline{U}$;
var **NewPeriod**: array[1...n] of integer;
**begin**
  $\overline{U}$ = 1;
  **for** (*i*=2 **to** n) **do**
  **begin**
    for (j=1 to i) do
    NewPeriod[j] = TaskPeriod[j] x $\left\lfloor \frac{TakPeriod[i]}{TaskPeriod[j]} \right\rfloor$;
    $\overline{U}' = \sum_{j=1}^{i-1} \frac{NewPeriod[j+1]-NewPeriod[j]}{NewPeriod[j]} + \frac{2NewPeriod[1]-NewPeriod[i]}{NewPeriod[i]}$
    if $\overline{U} > \overline{U}'$ then $\overline{U} = \overline{U}'$;
  **end**
  return ($\overline{U}$);
**end**

### *J. Chen, Mok, and Kuo Algorithms*

Chen, Mok, and Kuo [8] developed three polynomial-time algorithms with the aim to improve the performance of the *LL* and *HC* conditions.

On *Algorithm 1* (reproduced in Algorithm 4), a task set $\tau$ is transformed into another task set in which the ratio of any $T_i$ and $T_j$ (for all *i≠j)* is no larger than 2. A new utilization bound $U$ is calculated for the transformed task set, using Theorem 12 [8]. The task set $\tau$ can be feasibly scheduled under RM if its total utilization is less than or equal to $U$. The complexity of the *Algorithm 1* condition is $O(n^2)$.

**Theorem 12.** *Let $\tau$ = {$\tau_1$,$\tau_2$,..., $\tau_n$} be a set of periodic task. Let $\overrightarrow{T}$ be the array of the periods of the task set. If $T_1 < T_2 < \ldots < T_n < 2T_1$, then the utilization bound U for the task set is obtained when,*

$$C_i = T_{i+1} - T_i,\ 1 \leq i < n \tag{18}$$

$$C_n = 2T_1 - T_n \tag{19}$$

$$U = \sum_{i=1}^{n-1} \frac{T_{i+1} - T_i}{T_i} + \frac{2T_1 - T_n}{T_n} \tag{20}$$

**Example 11.** In this example, we will show the performance of the *Algorithm 1* condition. From the task set described in Table I, we compute the periods of the transformed task sets and their respective total utilization. Therefore, $\tau_1^{'}$ = [6, 8] and $U_1'$ = 0.8333; $\tau_2^{'}$ = [8,12,12] and $U_2'$ = 0.8333; $\tau_3^{'}$ = [12,15,16,16] and $U_3'$ = 0.8167; and $\tau_4^{'}$ = [48,48,48,48,48] and $U_4'$ = 1. Since the minimum of these utilization values is used as the schedulability bound, we have $U^{'}$ = 0.8167; therefore, tasks $\tau_1$, $\tau_2$, $\tau_3$, and $\tau_4$ can be schedulable. The *Algorithm 1* condition fails to identify task $\tau_5$ as schedulable because $U^{'} > U$ .

The second algorithm proposed by Chen, Mok, and Kuo [8] is named *Algorithm 2*, and it is not included in our comparison. This algorithm introduce a strategy to compute, with higher efficiency, the size of the harmonic base from a set of tasks, which is better than the bound obtained using the *HC* condition. The complexity of the *Algorithm 2* condition is $O(n^2)$. The *Algorithm 2* condition is based on Lemma 3 and Lemma 4 [8].

**Algorithm 5** *Algorithm 3* of Chen, Mok, and Kuo

**Input**: **TaskPeriod**[n] in non-decreasing order;
**Output**: Utilization bound $\overline{U}$;
var **TaskPeriod1**: array[1...n] of integer;
  [reduced task pattern]
  **TaskPeriod2**: array[1...n] of integer;
  [further reduced task pattern]
**begin**
  $\overline{U}$ = 1;
  **for** (*i*=1 **to** n) **do**
    **TaskPeriod1** = reduced task pattern
      of **TaskPeriod,** in non-decreasing order;
    **TaskPeriod2** = task pattern from **TaskPeriod1**
      with T = $\left\lfloor \frac{T_i}{T} \right\rfloor \cdot T$ for each element T,
      in non-decreasing order;
    $\overline{U}'$ = utilization factor calculated
      from **TaskPeriod2** using
      Theorem 12;
    if $\overline{U} > \overline{U}'$ then $\overline{U} = \overline{U}'$;
  **end for**
  return ($\overline{U}$);
**end**

Chen *et al.* proved in [8] that there is a smallest *m*, $1 \leq m \leq n$, and an array $\overrightarrow{T}^m = [T_1, T_2 \ldots, T_m]$ with total utilization $U^{'}$, reduced from $\overrightarrow{T}$ (the array of periods of the task set), such that $U^{'} \leq U$.

**Lemma 3.** *Let $U = U^m$. If $T_i$ divides $T_j$, $1 \leq i \leq j \leq m$, then U of $\overrightarrow{T}$ equals U of $\overrightarrow{T}' = [T_1, T_2, \ldots, T_{i-1}, T_{i+1}, \ldots, T_n]$.*

The definition of Lemma 4 is given next.

**Lemma 4.** *Let $U = U^m$. Consider $T_j$ and $T_k$ in $\overrightarrow{T}^m$, $t_m = t_jT_j + r_j = t_kT_k + r_k$. If $t_jT_j \leq t_kT_k$ and $e_k \leq \alpha_{jk}e_j$, then U is equal to the minimum utilization factor of all extreme task sets $\overrightarrow{T}' = [T_1, T_2, \ldots, T_{ki-1}, T_{k+1}, \ldots, T_n]$.*

The *Algorithm 3* condition is an improvement on the *Algorithm 2* condition. In this condition, $\overrightarrow{T}_{reduced}$ is a reduced array from $\overrightarrow{T}$ where some periods are deleted according to Lemma 3. Then, all the *extreme task sets* of $\overrightarrow{T}_{reduced}$ are generated. The utilization bound provided by the *Algorithm 3* condition is equal to the *minimum utilization factor* of all the extreme task sets of $\overrightarrow{T}_{reduced}$. The *Algorithm 3* condition is described in Algorithm 5. The complexity of the *Algorithm 3* condition is $O(n^3)$.

**Theorem 13.** *Let $\overrightarrow{T} = [T_1, T_2, \ldots, T_n]$. $U = \min_{i=1}^{n} U_i'$, $1 \leq i \leq n$, where $U_i'$ is the minimum utilization factor of all extreme task sets of the reduced array $\overrightarrow{T}' = [T_1, T_2, \ldots, T_i]$.*

**Example 12.** In this example, we will show the performance of the *Algorithm 3* condition. From the task set described in Table I, we have $\overrightarrow{T}$ = [3, 8, 12, 16, 48]. The reduced task set is $\overrightarrow{T}_{reduced}$ = [3, 8]. Transforming the reduced task set into a further reduced task sets by using the *Algorithm 3* and calculating its utilization using Theorem 12, we obtain the *minimum utilization factor* of the *extreme task set* as $U$ = 0.8333, which means that the task set comprised of tasks $\tau_1$, $\tau_2$, $\tau_3$, and $\tau_4$ is identified as schedulable. However, if task $\tau_5$

is included, the task set is not schedulable since $U'_5 > U$.

### K. CRMB Schedulability Condition

Lu *et al.* introduced the *Conditional RM Bound* (*CRMB*) schedulability condition in [28]. This condition extends the results provided by Lauzac *et al.* in [22] by using the relative period values in the task set.

Lauzac *et al.* showed in [22] that the schedulability bound used by the *R-Bound* condition is $\ln r$ + 2/*r* - 1 when the number of tasks approaches to infinity. If $z_1$ is the smallest period ratio in the task set and is defined as $z_1 = T_1/T_n$ = 1/*r*, the schedulability bound can be rewritten as $2z_1$ - $\ln z_1$ - 1.

Using the same worst-case scenario identified by Liu and Layland [26], where $T_i < 2T_1$, and defining $z_2$ as the largest period ratio in the task set (that is, $z_2 = T_{n-1}/T_n$), Lu *et al.* [28] improved the schedulability bound provided by Lauzac *et al.* to $2z_1$ + $1/z_2$ + $(\ln z_2 - \ln z_1)$ - 2.

In order to derive a schedulability bound for the case when some period values are less than or equal to $T_n/2$, Lu *et al.* defined the *virtual period* of a task [28].

**Definition 3.** (Virtual Period) *A virtual period of $\tau_i$, denoted by $v_i$, is the ready time of the critical job of $\tau_i$. That is,* $v_i = \left\lfloor \frac{T_n}{T_i} \right\rfloor T_i$

where a critical job is defined as:

**Definition 4.** *(Critical Jobs) The critical jobs are defined specifically at time $T_n$, the largest period in a task set. At $T_n$, the current jobs of all tasks, excluding $J_{1,n}$, are called the critical jobs of the system. Note that every task except $\tau_n$ has a critical job, and that the critical jobs are identified at time $T_n$.*

The smallest and the largest values among all virtual periods, $z_1$ and $z_2$, respectively, must be redefined in order to be used for the general case. These values are defined as follows:

$$z_1 = \min_{1 \le i \le n-1} \left\{ \frac{v_i}{T_n} \right\} \tag{21}$$

$$z_2 = \max_{1 \le i \le n-1} \left\{ \frac{v_i}{T_n} \right\} \tag{22}$$

The values of $z_1$ and $z_2$ are then used to derive the schedulability bound for the general case. The *CRMB* schedulability condition is formally defined in Theorem 14 [28] .

**Theorem 14.** *(CRMB Condition) Let $\tau$ be a task set with n periodic tasks. Suppose $z_1 = \min_{1 \le i \le n-1} \left\{ \frac{v_i}{T_n} \right\}$ and $z_2 = \max_{1 \le i \le n-1} \left\{ \frac{v_i}{T_n} \right\}$. Then $\tau$ is RM schedulable if $U \le$ CB($z_1$,$z_2$) = $2z_1 + \frac{1}{z_2} + (\ln z_2 - \ln z_1) - 2$.*

The *CRMB* schedulability condition achieves a higher schedulability bound if the difference between $z_1$ and $z_2$ is small. For instance, when the period values belong to a single harmonic chain, $z_1 = z_2 = 1$, the *CRMB* bound is 1. In [28], Lu *et al.* introduced a system design methodology to explore and adjust task periods using the *CRMB* schedulability condition, in order to achieve a higher utilization bound.

**Example 13.** In this example, we will show the performance of the *CRMB* schedulability condition using the task set of Table I. First, we need to obtain the *virtual period values* of the task set described in Table I. In this case, all tasks have virtual periods equal to their real periods. Next, we find the $z_1$ and $z_2$ values, where $z_1 = 1$ and $z_2 = 1$. Using Theorem 14, we obtain that CB($z_1$,$z_2$) = 1. Therefore, the task set $\tau$ described in Table I can be feasibly scheduled under RM according to the *CRMB* schedulability condition, since $U = 0.9375 \le CB(z_1, z_2) = 1$.

Using $z_1$ and $z_2$ values, Lu *et al.* derived, in [29], an exact test for RM on one processor. The complexity of the *CRMB* schedulability condition is $O(n)$.

### L. LP Schedulability Conditions

Lee *et al.* [23] introduced two linear programming formulations for calculating the utilization bounds for a given set of period options $(T_1, T_2, \ldots, T_k)$, where *k* is the number of period options $(1 \le k \le n)$ in the set, the periods of the tasks are $(T_1, T_2, \ldots, T_n)$, and $T_i < T_j$ for all *i*<*j*.

These schedulability conditions have a high computational complexity, and therefore, are not practical in online admission control. However, according to Lee *et al.* [23], these schedulability conditions are suitable for use in many practical real-time applications, in which the finite set of frequencies (periods) corresponds to the predetermined *QoS* options that the applications can choose, as in the audio and control applications. Thus, the utilization bound can be calculated offline using the finite set of periods, and then a *QoS* manager can use this bound online to determine the schedulability of the dynamically arriving task.

The first of the schedulability conditions, introduced in [23], is called *exact linear programming formulation* (*LpExact* condition). The second one, called *approximated linear programming formulation* (*LpApprox*), proposes a simpler formulation and is almost as accurate as the *exact linear programming formulation*. The experimental evaluation conducted in [23] showed that the *LpExact* condition outperforms the *LpApprox* condition. The complexity of the *LP* schedulability conditions is exponential [16].

*1) Exact Linear Programming Formulation (LpExact):* In order to calculate the tight bound $U^{*bound}$, the tight *level-i* bounds $U_i^{*bound}$ $(1 \le i \le k)$ need to be calculated first. $U^{*bound}$ provides the *system-level* bound, whereas the *level-i* bound $U_i^{*bound}$ provides the schedulability bound of only the *level-i* task $\tau_i$ that uses the period $T_i$. $U_i^{*bound}$ is called *tight level-i* bound, that is formally defined in Theorem 15 [23].

**Theorem 15.** *The minimum utilization $U^* = \sum_{j=1}^{i} \frac{C_j}{T_j}$ among those of all level-i barely schedulable task sets is the tight level-i bound $U_i^{*bound}$ .*

To calculate the *level-i* bound, all possible combinations of the execution times that make the task set *barely schedulable* are considered. A task set is *barely schedulable* if it is schedulable with the given execution time values, but a slight increment in any of its execution times makes the task set unschedulable. If a task set is *level-i* barely schedulable,

the *level-i* task $\tau_i$ is schedulable, and the processor is fully utilized by the *level-i* and the higher priority tasks during the interval $[0, T_i]$. It is possible to formulate a linear programming problem using a finite number of constraints. These constraints check the processor time demand only at the arrival times of the task instances (since the demand of the processor time changes only at those times) to determine the execution time values that make the *level-i* task set barely schedulable and to obtain its schedulability bound. The optimization problem to calculate the *level-i* bound $U_i^{*bound}$ is formulated in Theorem 16 [23].

**Theorem 16.** *The tight level-i bound $U_i^{*bound}$ is the solution for the following linear programming problem, where $T_j,\ 1 \leq j \leq i$ are fixed coefficients and $C_j,\ 1 \leq j \leq i$ are free variables,*

$$U_i^{*bound} = Minimize \sum_{j=1}^{i} \frac{C_j}{T_j} \tag{23}$$

*Subject to*

$$\sum_{j=1}^{i-1} \left\lceil \frac{T_i}{T_j} \right\rceil C_j + C_i = T_i \tag{24}$$

$$\sum_{j=1}^{i-1} \left\lceil \frac{t_a}{T_j} \right\rceil C_j + C_i \geq t_a,\ 1 \leq a \leq M \tag{25}$$

*where $t_a,\ 1 \leq a \leq M$ are the series of all the arrival instants of the higher priority tasks in [0,$T_i$].*

Theorem 16 assures only the schedulability of the *level-i* task $\tau_i$. The tight *system-level* bound $U^{*bound}$ is given by the minimum of the *level-i* bounds, as stated in Theorem 17 [23].

**Theorem 17.** *(LpExact Condition) The minimum of the tight level-i $(1 \leq i \leq k)$ bounds, that is, $\min_{i=1}^{k} U_i^{*bound}$, is the tight (largest sufficient) system-level $U^{*bound}$.*

**Example 14.** In this example, we will show the performance of the *LpExact* schedulability condition using the task set of Table I. To use these conditions, we first sort the tasks by a non-decreasing order of their period values. To illustrate the *LpExact* condition, Fig. 6 shows the linear programming problem formulation for $U_3^{*bound}$. Using Theorem 16 we obtain all *level-i* bounds, that is, $U_1^{*bound}$= 1, $U_2^{*bound}$= 0.9167, $U_3^{*bound}$= 0.875, $U_4^{*bound}$= 0.875, and $U_5^{*bound}$= 1. According to Theorem 17, the *system-level* bound $U^{*bound}$ is equal to 0.875. Since the total utilization of $\tau$ is greater than $U^{*bound}$, the *LpExact* condition fails to identify the task set $\tau$ as schedulable under RM.

*2) Approximate Linear Programming Formulation (LpApprox):* The main drawback of the *exact linear programming formulation* is its complexity. With a large number of period options, there can be a very large number of arrival instants resulting in a huge number of constraints.

In the *LpExact* condition, most of the constraints are used to check the processor time demand at all arrival instants to avoid the potential idle times. However, the idle times tend to occur late in the interval [0,$T_i$] when the processor is heavily loaded. This means that checking only at the last arrival of each task before *t*=$T_i$ can avoid most of the potential idle times. The *LpApprox* condition just checks those last arrival times, resulting in a simpler linear programming formulation. This approximated formulation has only one constraint at each level and thus, the total number of constraints used to calculate the *level-i* bound $U_i^{bound}$ is *i*, as can be observed in Theorem 18 [23].

**Minimize**

$U_3^{*bound}$ = **C1**/3 + **C2**/8 + **C3**/12;

**Subject to**

4 **C1** + 2 **C2** + 1 **C3** = 12; (1)
1 **C1** + 1 **C2** + 1 **C3** >= 3; (2)
2 **C1** + 1 **C2** + 1 **C3** >= 6; (3)
3 **C1** + 2 **C2** + 1 **C3** >= 9; (4)
3 **C1** + 1 **C2** + 1 **C3** >= 8; (5)

Figure 6. *LP* Formulation Problem for $U_3^{*bound}$

**Theorem 18.** *The solution for the following linear programming problem, $U_i^{bound}$, is a sufficient level-i bound.*

$$U_i^{bound} = Minimize \sum_{j=1}^{i} \frac{C_j}{T_j} \tag{26}$$

*Subject to*

$$\sum_{j=1}^{i-1} \left\lceil \frac{T_i}{T_j} \right\rceil C_j + C_i = T_i \tag{27}$$

$$\sum_{j=1}^{i-1} \left\lceil \frac{\left\lfloor \frac{T_i}{T_a} \right\rfloor T_a}{T_j} \right\rceil C_j + C_i \geq \left\lfloor \frac{T_i}{T_a} \right\rfloor T_a,\ 1 \leq a \leq M \tag{28}$$

*where $\left\lfloor \frac{T_i}{T_a} \right\rfloor T_a,\ 1 \leq a \leq i-1$ is the last arrival instant of task $\tau_a$ before $T_i$.*

As in the tight *system-level*, a sufficient system-level bound can be found by taking the minimum of the *level-i* bounds, as expressed in Theorem 19 [23].

**Theorem 19.** *(LpApprox Condition) The minimum of the sufficient level-i $(1 \leq i \leq k)$ bounds, that is, $\min_{i=1}^{k} U_i^{bound}$, is the sufficient system-level $U^{bound}$.*

**Example 15.** In this example, we will show the performance of the *LpApprox* schedulability condition using the task set shown in Table I. Fig. 6 shows the linear programming problem formulation for $U_3^{bound}$. However, unlike the exact formulation that uses all the constraints shown in Fig. 6, the *LpApprox* condition only uses the constraints 1, 4, and 5. Using Theorem 16, we obtain all *level-i* bounds, that is, $U_1^{bound}$= 1, $U_2^{bound}$= 0.9167, $U_3^{bound}$= 0.875, $U_4^{bound}$= 0.875, and $U_5^{bound}$= 1. According to Theorem 18, the system-level bound $U^{bound}$ is equal to 0.875. Since the *total utilization* of $\tau$ is greater than $U^{bound}$, the *LpApprox* condition fails to identify the task set $\tau$ as schedulable under RM.

| *Condition* | *Utilization Bound* | *Complexity* |
|---|---|---|
| *LL* | $U \leq n(2^{\frac{1}{n}} - 1)$ | *O(n)* |
| *IP* | $u_n \leq 2\left(1 + \frac{U_{n-1}}{(n-1)}\right)^{-(n-1)} - 1$ | *O(n log n)* |
| *PO* | $U \leq (n-1)(2^{\beta/(n-1)} - 1) + 2^{1-\beta} - 1$ | *O(n log n)* |
| *UO, HB* | $U \leq 2\left[\prod_{i=1}^{n-1}(1+u_i)\right]^{-1} - 1$ | *O(n)* |
| *T-Bound* | $U \leq \sum_{i=1}^{n-1}\left[\frac{T'_{i+1}}{T'_i}\right] + 2\frac{T'_i}{T'_n} - n$ | *O(n log n)* |
| *R-Bound* | $U \leq (n-1)(r^{\frac{1}{(n-1)}} - 1) + \left(\frac{2}{r}\right) - 1$ | *O(n log n)* |
| *HC* | $U \leq k\,(2^{\frac{1}{k}} - 1)$ | $O(n^{\frac{5}{2}})$ |
| *Root* | $U \leq R\,(2^{\frac{1}{R}} - 1)$ | $O(n^2)$ |
| *Sr* | *non closed-form* | *O(n log n)* |
| *DCT* | *non closed-form* | $O(n^2)$ |
| *Alg 1* | *non closed-form* | $O(n^2)$ |
| *Alg 2* | *non closed-form* | $O(n^2)$ |
| *Alg 3* | *non closed-form* | $O(n^3)$ |
| *CRMB* | $U \leq 2z_1 + \frac{1}{z_2} + (\ln z_2 - \ln z_1) - 2$ | *O(n)* |
| *LpExact* | *Linear programming* | *exponential* |
| *LpApprox* | *Linear programming* | *exponential* |

Table VII
COMPARISON OF THE RM INEXACT SCHEDULABILITY CONDITIONS

### M. Characteristics of the Schedulability Conditions

A summary of the inexact schedulability conditions for RM on one processor, discussed in this section, is shown in Table VII. It can be noted that the *LP* and *Algorithm 3* schedulability conditions have the highest complexity, whereas the *LL, UO,* and *CRMB* schedulability conditions have the lowest complexity.

## V. EVALUATION RESULTS

To evaluate the performance of the inexact schedulability conditions discussed in this paper, we tested every condition using a sample *s* of task sets that are schedulable under rate monotonic. We define the *acceptance ratio* $\rho$ of a schedulability condition *SC* for a sample *s*, as the ratio of the number of tasks identified as schedulable by the schedulability condition and the total number of task sets:

$$\rho_s(SC) = \frac{number\ of\ task\ sets\ accepted\ by\ SC}{total\ number\ of\ task\ sets} * 100 \qquad (29)$$

Since all task sets in *s* are schedulable under RM, an exact condition will have an acceptance ratio equal to 100. If $\rho_s(SC)$ approaches 100, then the performance of *SC* approaches the performance of the exact test for *s*.

With the purpose of evaluating the performance of the schedulability conditions under different characteristics of the task sets, we conducted our experiments using four different schemes of generation of the task sets in *s*.

The solution of the linear programming problems formulated by the *LpExact* condition was obtained using the *lp_solve* package [4].

### A. Performance as a function of the number of tasks

The goal of this experiment was to evaluate the performance of the schedulability conditions as a function of the number of tasks. We generated eleven samples of task sets denoted as $N_2, N_3, \ldots, N_{12}$. Each sample $N_m$ was conformed by 1,000 sets of *m* tasks. The total utilization of each task set and the periods of the tasks were uniformly distributed in the range [0.7, 0.95] and [100, 500], respectively. The maximum utilization of each task, denoted by $\alpha$, followed a uniform distribution in the range [0.01, 0.3]. The execution times of the tasks were generated with values $1 \leq C_i \leq \alpha T_i$. Figs. 7, 8, and 9 show the acceptance ratios obtained as a function of the number of tasks for these conditions in their respective groups.

Fig. 7 shows the acceptance ratios obtained for the *closed-form non-period-aware* schedulability conditions. For these conditions, the performance decreases rapidly as the number of tasks increases. For each number of tasks, the acceptance ratios always satisfy the relation $\rho_N(UO) > \rho_N(IP) > \rho_N(LL)$. This result is a consequence of the amount of timing information of the tasks used by each of the schedulability conditions. It is important to note that the improvement achieved by these conditions with respect to the *LL* condition is marginal (that is, smaller than 8%).

Fig. 8 shows the acceptance ratios obtained for the *closed-form period-aware* schedulability conditions where we also included the *UO* and *LL* conditions.

From this experiment, it can be observed that the *PO* and the *CRMB* are the conditions with better performance for small number of tasks (*m*<4). However, for *m*>4, only the *PO* and the *UO* conditions showed some performance improvement over the *LL* condition (less than 4%). These poor results can be explained by the fact that this experiment was designed without considering any relationship among the periods of the tasks, and since these conditions include the period in their analysis, it is clear that they cannot take advantage of this extra information.

Fig. 9 shows the acceptance ratios obtained for the *non-closed-form period-aware* schedulability conditions, where we also included the *PO* condition and the *LL* condition. Due to the differences in the approaches used to derive their schedulability bounds and their resulting behaviors, these schedulability conditions can be differentiated as follows:

- *Schedulability condition based on linear programming*. The *LpExact* schedulability condition has the second best acceptance ratio for very small task sets (for $m < 4$, only lower than that of the *DCT* condition). For the larger task sets (*that is,* $m \geq 4$), its performance decreases faster than the performance of the *DCT* and the *Algorithm 1* conditions, showing the third best acceptance ratio among all *non-closed-form period-aware* conditions. However, it is important to recall that the computational complexity of this condition is exponential.

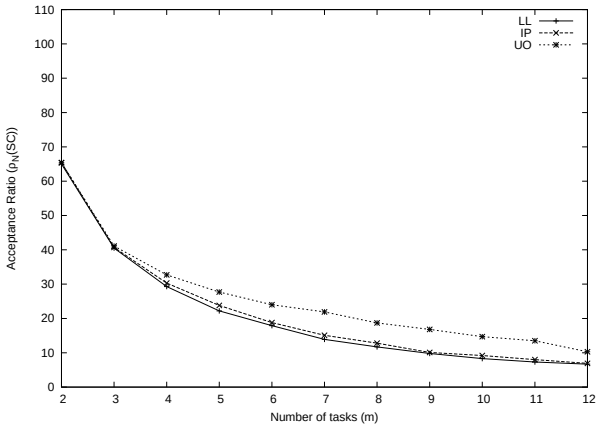


Figure 7. Acceptance ratio of the *non-period-aware* schedulability conditions

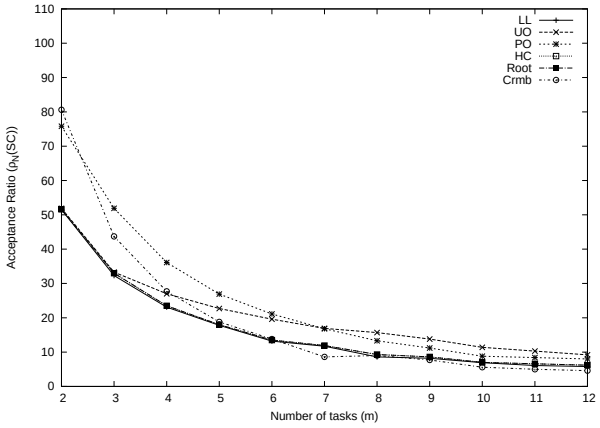


Figure 8. Acceptance ratio of the *closed-form* schedulability conditions

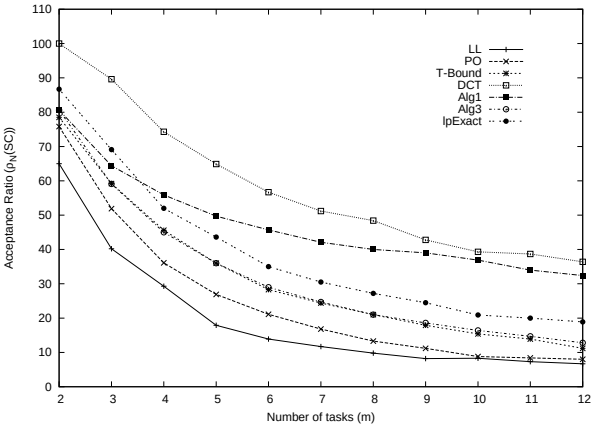


Figure 9. Acceptance ratio of the *non-closed-form* schedulability conditions

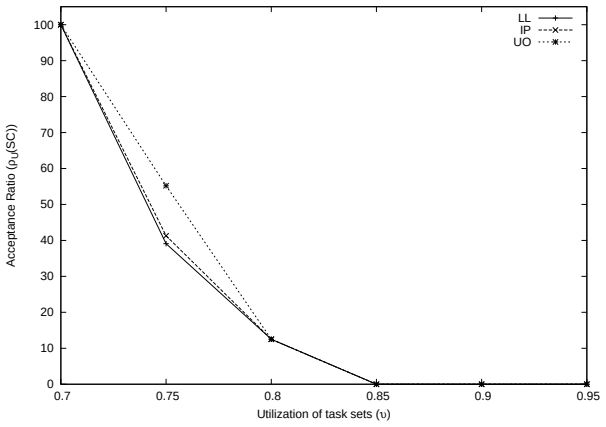


Figure 10. Acceptance ratio of the *non-period-aware* schedulability conditions

- *Schedulability conditions based on the transformation of the original task set* (*T-Bound*, *Algorithm 1*, *Algorithm 3* and *DCT*). It can be noted from Fig. 9 that for every number of tasks, the acceptance ratio of these schedulability conditions was substantially better than the acceptance ratio of the *closed-form* schedulability conditions. The *DCT* condition showed the best performance among all *non-closed-form period-aware* conditions for *m*<6, with a clear improvement over the performance of the *Algorithm 1* condition. However, for large number of tasks, their performance tended to be similar. On the other hand, the performance of the *Algorithm 1* was always better than that of the *Algorithm 3* and the *T-Bound* conditions. The *Algorithm 3* and the *T-Bound* conditions always obtained very similar performances, at least 6% better than the *LL* condition for large number of tasks.

### *B. Performance as a function of the total utilization of task sets*

The goal of this experiment was to evaluate the performance of the schedulability conditions as a function of the total utilization of the task sets. We generated several samples of task sets denoted as $U_\upsilon$, where the total utilization $\upsilon$ of each sample was set to 0.7, 0.75, 0.8, 0.85, 0.9, and 0.95. The maximum utilization of each task, denoted by $\alpha$, followed a uniform distribution in the range [0.01, 0.3]. Each sample $U_\upsilon$ was conformed by 1,000 task sets. The periods of the tasks were uniformly distributed in the range [100, 500] and the execution times of the tasks were generated with values $1 \leq C_i \leq \alpha T_i$. The number of tasks was uniformly distributed in the range [2,9]. Figs. 10, 11, and 12 show the acceptance ratios obtained as a function of the utilization of the task sets for the schedulability conditions.

Fig. 10 shows the acceptance ratios obtained for the *closed-form non-period-aware* schedulability conditions. We can observe that the acceptance ratios of these conditions satisfy the relation $\rho_U(UO) > \rho_U(IP) > \rho_U(LL)$. The performance

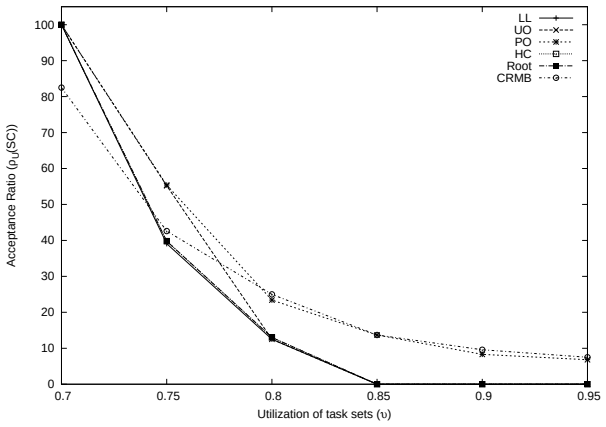


Figure 11. Acceptance ratio of the *closed-form period-aware* schedulability conditions

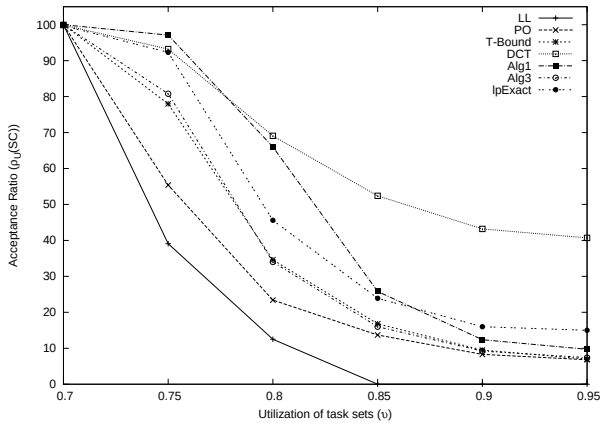


Figure 12. Acceptance ratio of the *non-closed-form period-aware* schedulability conditions

of the*UO* is clearly better than the *IP* and the *LL* conditions in the range $0.7 < \upsilon < 0.8$.

Fig. 11 shows the acceptance ratios obtained for the *closed-form period-aware* schedulability conditions, where we also included the *UO* and the *LL* conditions. We can observe from Fig. 11 that the *PO* condition showed the best performance among the *closed-form period-aware* conditions for $\upsilon < 0.8$, improving upon the *LL* condition for all values of $\upsilon$. For $\upsilon \geq 0.75$, the *CRMB* condition showed an acceptance ratio similar or better than the *PO* condition. However, the acceptance ratio of the *CRMB* condition was poorer than that of the *LL* condition for $\upsilon = 0.7$. The *HC* and the *Root* conditions showed similar acceptance ratios as that of the *LL* condition for all the values of $\upsilon$.

Fig. 12 shows the acceptance ratios obtained for the *non-closed-form period-aware* schedulability conditions, where we also included the *PO* and the *LL* conditions. We can observe from Fig. 12 that all *non-closed-form* conditions showed a significant performance improvement with respect to the *closed-form* conditions, and that the *DCT* condition yielded the best acceptance ratio among all the *non-closed-form* period-aware conditions. When compared with the other conditions, the *DCT* condition showed an increasingly better performance when the utilization of the task sets increased.

The *Algorithm 1* and the *LpExact* conditions also showed a good performance, even improving on the *DCT* condition for $\upsilon \leq 0.75$. However, their performance decreased rapidly as the utilization of the task sets increased.

The *Algorithm 3* and the *T-Bound* conditions showed the worst performance among all the *non-closed-form* conditions, with results close to the *PO* condition for $\upsilon \geq 0.85$.

### C. Performance as a function of the period ratio

The goal of this experiment was to evaluate the performance of the schedulability conditions as a function of the *period ratio*. We define the *period ratio* as the ratio of the maximum and minimum period values in a task set. We generated several samples $R_\lambda$, where $\lambda$ is a constant with values in the range [1, 8] used to derive the periods of the tasks. For each task set, we randomly generated an initial period value $T_1$ in the range [100, 300]. After $T_1$ was generated, the periods of the remaining tasks were generated in the range $T_1 \leq T_i \leq \lambda * T_1$. The total utilization of every sample and the maximum utilization $\alpha$ of each task followed a uniform distribution in the range [0.7, 0.95] and [0.01, 0.3], respectively. The execution times of the tasks were generated with values in the range $1 \leq C_i \leq \alpha T_i$. The number of tasks was uniformly distributed in the range [2,9]. Every sample $R_\lambda$ was composed of 1,000 task sets. Figs. 13, 14, and 15 show the performance as a function of the period ratio for the schedulability conditions.

Fig. 13 shows the acceptance ratios obtained for the *closed-form non-period-aware* schedulability conditions. It can be noted that for all the values of $\lambda$, the acceptance ratios satisfy the relation $\rho_R(UO) > \rho_R(IP) > \rho_R(LL)$. The *UO* condition showed the best acceptance ratio among these conditions, with a small performance improvement with respect to the *IP* and the *LL* conditions, whereas these two conditions (*IP* and *LL*) yielded similar acceptance ratios. It is important to note that all *closed-form non-period-aware* conditions showed a constant acceptance ratio for all the values of $\lambda$.

Fig. 14 shows the acceptance ratios obtained for the *closed-form period-aware* schedulability conditions, where we also included the *UO* and the *LL* conditions. We can observe that all the *closed-form period-aware* schedulability conditions yielded an acceptance ratio of 100 for $\lambda = 1$. In addition, all of them decreased their performance sharply for the interval $1 < \lambda < 2$, and for $\lambda > 2$, their performance was constant with the small variations.

The *PO* condition was the best or second best of all the *closed-form period-aware* conditions for all values of $\lambda$. The *CRMB* condition showed the best performance among all the *closed-form period-aware* conditions for $\lambda \leq 1.5$. However, its performance was the worst of all the *closed-form period-aware* conditions for $\lambda = 2$, but improved for $\lambda > 2$, being

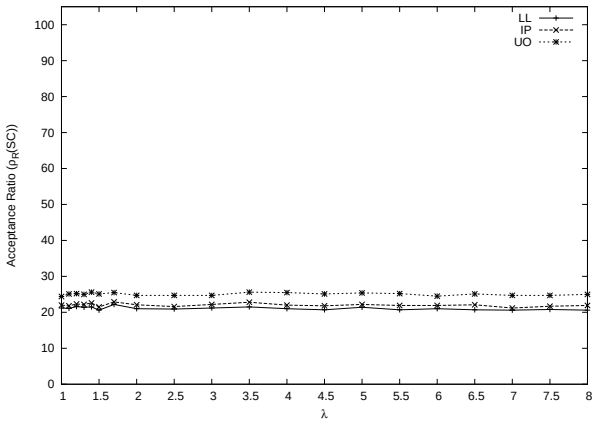


Figure 13. Acceptance ratio of the *closed-form non-period-aware* schedulability conditions

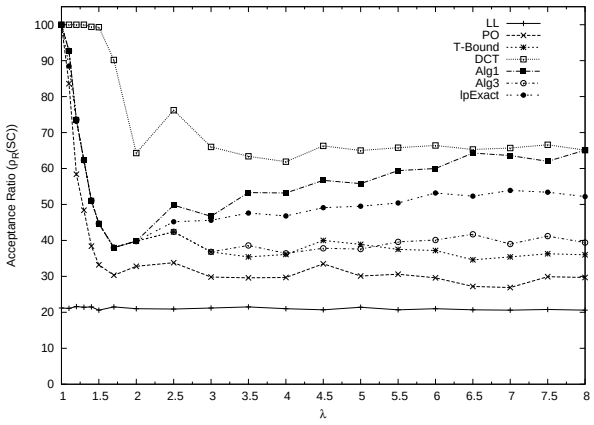


Figure 15. Acceptance ratio of the *non-closed-form* schedulability conditions

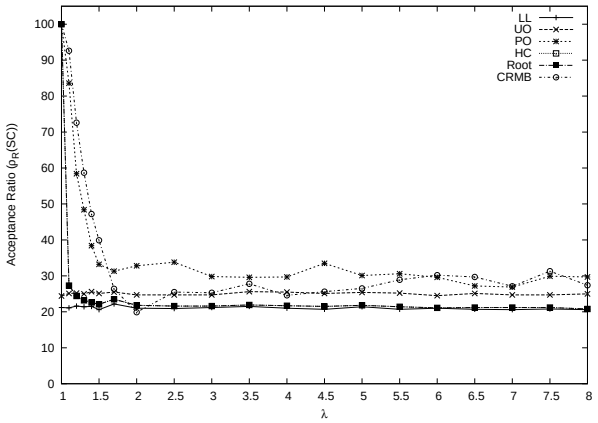


Figure 14. Acceptance ratio of the *closed-form period-aware* schedulability conditions

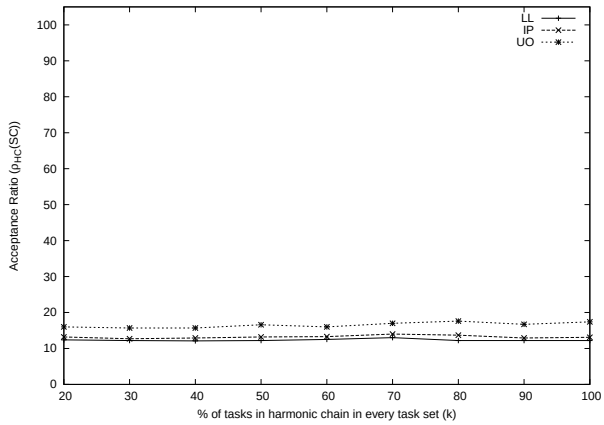


Figure 16. Acceptance ratio of the *closed-form non-period-aware* schedulability conditions

among the best conditions for $\lambda \geq 4.5$. Finally, the *Root* and the *HC* conditions showed a performance similar to that of the *LL* condition for all values of $\lambda$.

Fig. 15 shows the acceptance ratios obtained for the *non closed-form* schedulability conditions, where we also included the *PO* and *LL* conditions. From Fig. 15, we can observe that all *non-closed-form* conditions clearly outperformed the remaining conditions.

The *DCT* condition showed the best acceptance ratio among the *non-closed-form* schedulability conditions for all the values of $\lambda$. It should be noted that for $\lambda < 1.5$, the acceptance ratio of the *DCT* condition is almost equal to 100. On the other hand, for$\lambda > 2$, its acceptance ratio is almost constant.

The *Algorithm 1*, the *Algorithm 3*, and the *T-Bound* conditions showed a fairly good performance for small $\lambda$ values ($\lambda \leq 1.2$). These three conditions showed identical acceptance ratios for $\lambda < 2$. This can be explained by the fact that they transform the original task set into another task set where all period values satisfy the relation $T_{max}/T_{min} < 2$. However, for $\lambda > 2$, $\rho_R$(*Algorithm 1*) > $\rho_R$(*Algorithm 3*) $\geq$ $\rho_R$(*T-Bound*), and for $\lambda > 1.5$, the *Algorithm 1* condition showed the second best acceptance ratio among all the conditions.

The *LpExact* condition showed a good performance for the small values of $\lambda$ ($\lambda \leq 1.2$), whereas for $\lambda > 1.5$, its acceptance ratio was the third best among all the schedulability conditions.

It is important to note that the acceptance ratios of all the conditions remain stable for $\lambda > 2$.

### *D. Performance as a function of tasks in the harmonic chain*

The objective of this experiment was to evaluate the performance of the schedulability conditions as a function of the percentage of tasks that are part of a harmonic chain. We generated nine samples of task sets $HC_k$ conformed by 1,000 task sets, where *k* is a value in the range [20, 100] that denotes

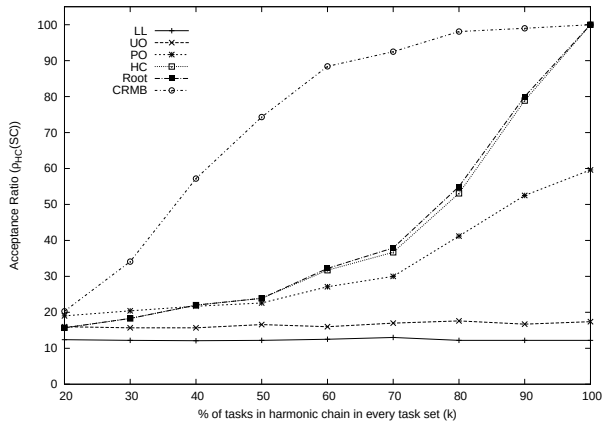


Figure 17. Acceptance ratio of the *closed-form period-aware* schedulability conditions

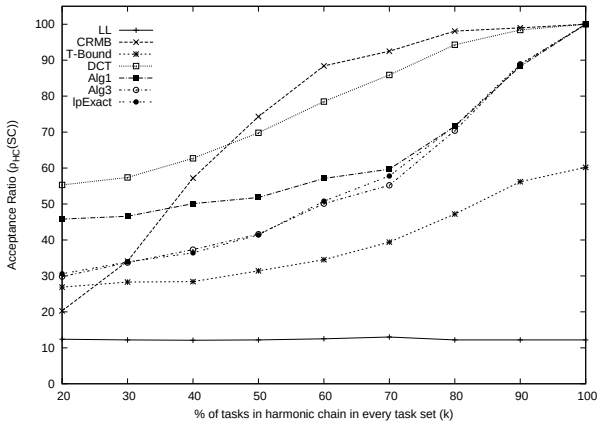


Figure 18. Acceptance ratio of the *non-closed-form period-aware* schedulability conditions

the percentage of tasks that are part of the harmonic chain. Only one harmonic chain in every task set was generated. The utilization of every sample and the maximum utilization $\alpha$ of each task was uniformly distributed in the range [0.7, 0.95] and [0.01, 0.30], respectively. The execution times of the tasks were generated with values $1 \leq C_i \leq \alpha T_i$. The number of tasks was uniformly distributed in the range [5,9].

The period values of the tasks were generated as follows. The initial period $T_1$ was obtained using a uniform distribution in the range [20, 100]. Once $T_1$ was derived, the period of each task was generated using $T_i = T_{i-1} * f$, where $f$ was randomly generated in the range [2,3], until the defined percentage of tasks in the harmonic chain was reached. If this percentage was $k$<100%, the remaining period values were generated such that they did not belong to the harmonic chain. Figs. 16, 17, and 18 show the obtained acceptance ratios as a function of the percentage of tasks that are part of an harmonic chain.

Fig. 16 shows the acceptance ratios obtained for the *closed-form non-period-aware* schedulability conditions. We can observe that their acceptance ratios satisfy the relation $\rho_{HC}(UO)$ > $\rho_{HC}(IP)$ > $\rho_{HC}(LL)$ for every value of *k*. The *UO* condition is slightly better than the *IP* and the *LL* conditions, whereas the acceptance ratio of the *IP* and the *LL* conditions are very close to each other. From these results, it is clear that none of these conditions benefit from including tasks that are part of a harmonic chain.

Fig. 17 shows the acceptance ratios obtained for the *closed-form period-aware* schedulability conditions, where we also included the *UO* and the *LL* conditions. We can observe that in most cases, their acceptance ratios satisfy the relation $\rho_{HC}$(*CRMB*) > $\rho_{HC}$(PO) > $\rho_{HC}$(*Root*) > $\rho_{HC}$(*HC*). It can be observed that the *CRMB* condition had an acceptance ratio significantly better than that of the remaining *closed-form period-aware* conditions, showing an excellent performance for $k \geq 60$. As discussed previously, the acceptance ratio of the *CRMB* condition was equal to 100 when all the tasks were in a harmonic chain. The *HC* and the *Root* conditions showed a very similar performance, only lower than that of the *CRMB* condition. The *PO* condition showed a performance similar to the *HC* and the *Root* conditions for values of $k \leq 50$, but much lower than the *HC* and the *Root* conditions for higher values of *k*.

Fig. 18 shows the acceptance ratios obtained for the *non-closed-form period-aware* schedulability conditions, where we also included the *CRMB* and the *LL* conditions. We can observe that the *DCT* condition showed the best performance among the *non-closed-form period-aware* conditions. Nevertheless, its performance was not as good as the performance of *CRMB* condition for $k \geq 50$.

The *Algorithm 1*, the *Algorithm 3*, and the *LpExact* conditions showed a similar performance for $k \geq 60$. However, the *Algorithm 3* and the *LpExact* conditions showed a lower acceptance ratio for values of *k* smaller than 60. The *T-Bound* condition showed the worst performance among these schedulability conditions. It is interesting to note that the *DCT, Algorithm 1, Algorithm 3, LpExact*, and *CRMB* conditions yielded an acceptance ratio equal or close to 100 for *k*=100.

### *E. Comparison of Performances of the Schedulability Conditions*

A comparison of the relative performance of the inexact schedulability conditions for RM on one processor is shown in Table VIII. The aim of this comparison is to summarize the results of the experiments conducted in this section. Designers of real-time applications can use the comparison shown in Table VIII to determine which schedulability condition may be used in certain situations, taking into account the characteristics of the task set.

It can be noted that the *non-closed-form period-aware* conditions yield better performance than the *closed-form* conditions.

## VI. Conclusions and Future Work

Many real-time applications demand efficient and low-cost schedulability tests for online admission control. In this paper,

| *Characteristics of task sets* | *Performance* | | |
|---|---|---|---|
| | *Good* | *Average* | *Poor* |
| *Small number of tasks ($m \leq 4$)* | *DCT* | *PO, CRMB, T-Bound, Alg1, Alg3, LpExact* | *LL, IP, UO, HC, Root* |
| *Large number of tasks ($m > 7$)* | *DCT, Alg1* | *Alg3, T-Bound, LpExact* | *LL, IP, UO, HC, Root, PO, CRMB* |
| *Small utilization of task sets ($U \leq 0.8$)* | *Alg1, DCT, LpExact* | *Alg3, T-Bound,* | *LL, IP, UO, PO, Root, HC, CRMB* |
| *Large utilization of task sets ($U > 0.8$)* | *DCT* | *Alg1, LpExact* | *LL, IP, UO, HC, Root PO, CRMB, Alg3* |
| *Small period ratio ($\lambda \leq 1.5$)* | *DCT* | *CRMB, Alg1, Alg3, T-Bound, LpExact* | *LL, IP, UO, HC, Root, PO,* |
| *Large period ratio ($\lambda > 1.5$)* | *DCT, Alg1 LpExact* | *Alg3, T-Bound, CRMB, PO* | *LL, IP, UO,HC, Root* |
| *All tasks in a single hc* | *DCT, CRMB, Alg3 Alg1, HC, Root , LpExact* | *PO, T-Bound,* | *LL, IP, UO,* |
| *Large number of tasks in hc ($k \geq 80\%$)* | *CRMB, DCT* | *Alg3, Alg1, LpExact, HC, Root, T-Bound, PO,* | *LL, IP, UO,* |
| *Small number of tasks in hc ($80\% > k \geq 20\%$)* | *DCT, Alg1* | *Alg3, T-Bound, LpExact* | *HC, Root ,PO, LL IP, UO, CRMB* |

Table VIII
RELATIVE PERFORMANCE OF THE RM INEXACT SCHEDULABILITY CONDITIONS

we surveyed the best-known exact and inexact schedulability conditions for Rate Monotonic executing on one processor. Extensive simulation experiments were conducted to evaluate the performance and computational complexity of the inexact schedulability tests. In our simulation experiments, the schedulability tests were evaluated for different number of tasks, utilization factors, and different period ratios. Additional experiments were conducted considering task sets with harmonics chains.

The comparative analysis done in this paper showed that for all the experiments conducted, the schedulability conditions using the *non-closed-forms* schedulability tests derive a better performance than those that use the *closed-forms* schedulability tests.

Among all the *non-closed-form* schedulability conditions, we observed that, in general, the *DCT* condition showed the best performance. This performance can be explained by the fact that the *DCT* condition transforms the period set into another period set where all the tasks belong to a single harmonic chain.

We believe that the decision of choosing one schedulability test over another for a particular real-time application should not depend only on its performance; it should also be take into consideration the characteristics of the tasks and their computational complexity.

As part of our future research, we plan to extend this study to include schedulability tests for aperiodic, resource-sharing tasks, and multiple processors.

## REFERENCES


[1] Atdelzater, T.F., Atkins, E.M., and Shin, K.G. Qos negotiation in real-time systems and its application to automated flight control. *IEEE Transactions on Computers*, 49(11):1170–1183, Nov 2000.

[2] Audsley, N., Burns, A., Richardson, M., Tindell, K., and Wellings, A.J. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8:284–292, Dec 1993.

[3] Banerjea, A., Ferrari, D., and Mah, B.A. Moran, M. The tenet real-time protocol suite: Design, implementation, and experiences. *IEEE/ACM Transactions on Networking*, 4(8):1–10, 1994.

[4] Berkelaar, Michael, Eikland, Kjell, and Notebaert, Peter. lp_solve 5.5. http://lpsolve.sourceforge.net/5.5, May 2004.

[5] Bini, E., Buttazzo, G.C., and Buttazzo, G.M. Rate monotonic analysis: The hyperbolic bound. *IEEE Transactions on Computers*, 52(7):933–942, Jul 2003.

[6] Bini, E. and Buttazzo, Giorgio C. Schedulability analysis of periodic fixed priority systems. *IEEE Transactions on Computers*, 53(11):1462–1473, Nov 2004.

[7] Burchard, A., Liebeherr, J., Oh, Y., and Son, S.H. New strategies for assigning real-time tasks to multiprocessor systems. *IEEE Transactions on Computers*, 44(12):1429–1442, Dec 1995.

[8] Chen, D., Mok, R., and Kuo, T. Utilization bound revisited. *IEEE Transactions on Computers*, 53(3):351–361, March 2003.

[9] Chen, H., Tjaden, B.C., Welch, L.R., Bruggeman, C., Tong, L., and Pfarr, B. Monitoring network QoS in a dynamic real-time system. In *Proceedings of the 16th IEEE International Parallel and Distributed Processing Symposium*, pages 93–99, Fort Lauderdale, Florida, USA, 2002. IEEE Computer Society.

[10] Cheng, A.M. K. and Rao, S.M. Real-time traffic scheduling and routing in packet-switched networks using a least-laxity-first strategy. *Journal of VLSI Signal Processing Systems*, 34(1/2):139–148, 2003.

[11] Davis, R.I., Zabos, A., and Burns, A. Efficient exact schedulability tests for fixed priority real-time systems. *IEEE Transactions on Computers*, 57(9):1261–1276, Sep 2008.

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

[13] C.-C.J. Han. A better polynomial-time schedulability test for real-time multiframe tasks. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, pages 104–113, Madrid, Spain, Dec 1998. IEEE Computer Society.

[14] Han, C.C. and Tyan, H.Y. A better polynomial-time schedulability test for real-time fixed-priority scheduling algorithms. In *Proceedings of 19th IEEE Real-Time Systems Symposium*, pages 36–45, San Francisco, CA, USA, Dec 1997. IEEE Computer Society.

[15] Joseph, M. and Pandya, P. Finding response times in a real-time system. *British Computer Society Computer Journal*, 29(5):390–395, Oct 1986.

[16] Klee, V. and Minty, G.J. *How Good is the Simplex Algorithm?, in Inequalities III*. O. Shisha (ed), Academic Press Inc, New York, April 1997.

[17] Kuo, T. and Lin, K. Efficient on-line schedulability tests for priority driven real-time systems. In *In Proceedings of 6th IEEE Real-Time Technology and Applications Symposium*, pages 4–14, Washington D.C., USA, May-Jun 2000.

[18] Kuo, T. and Mok, A. Load adjustment in adaptive real-time systems. In *Proceedings of the 12th IEEE Real-Time Systems Symposium*, pages 160–170, San Antonio, TX, USA, Dec 1991. IEEE Computer Society.

[19] Kuo, T-W. and Li, C-H. A fixed-priority-driven open environment for real-time applications. In *Proceedings of the 20th IEEE Real-Time Systems Symposium*, pages 256–267, Phoenix, AZ, USA, Dec 1999. IEEE Computer Society.

[20] Kuo, T-W., Li, C-H., and Wang, Y-C. An open real-time environment for parallel and distributed systems. In *Proceedings of the 20th International Conference on Distributed Computing Systems*, pages 206–213, Taipei, Taiwan, Apr 2000.

[21] Lauzac, S., Melhem, R., and Mosse, D. An efficient rms admission control and its application to multiprocessor scheduling. In *Proceedings of the IEEE 1st Merged International Symposium on Parallel and Distributed Processing*, pages 511–518, Orlando, Fl, USA, April 1998.

[22] Lauzac, S., Melhem, R., and Mosse, D. An improved rate-monotonic admission control and its applications. *IEEE Transactions on Computers*, 52(2):337–350, March 2003.

[23] Lee, C-G., Sha, L., and Peddi, A. Enhanced utilization bounds for QoS management. *IEEE Transactions on Computers*, 53(2):187–200, Feb 2004.

[24] Lehoczky, J.P., Sha, L., and Ding, Y. The rate-monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 166–171, Santa Monica, CA, USA, Dec 1989. IEEE Computer Society.

[25] Leung, J.Y. and Whitehead, J. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation (Netherlands)*, 4(2):237–250, Dec 1982.

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

[27] Liu, J.W.S. *Real-Time Systems*. Prentice Hall, 2000.

[28] Lu, W.-C., Lin, K.J., Wei, H.-W., and Shih, W.-K. Rate monotonic schedulability tests using period-dependent conditions. *Real-Time Systems*, 37(2):123–138, Nov 2007.

[29] Lu, W.C., Lin, K.J., Wei, H.W., and Shih, W.K. Period-dependent initial values for exact schedulability test of rate monotonic systems. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, pages 1–8, Long Beach, California, USA, March 2007.

[30] Miller, L.K. and Cheng, A.M.K. Admission of high priority real-time calls in an atm network viabandwidth reallocation and dynamic rerouting of active channels. In *Proceedings of the 21st Real-Time Systems Symposium*, pages 249–258, Orlando, Fl., Nov 2000. IEEE Computer Society.

[31] Oh, Y. and Son, S.H. Fixed-priority scheduling of periodic tasks on multiprocessor systems. Technical Report CS-95-16, Univ. of Virginia. Dept. of Computer Science, March 1995.

[32] Park, D.W., Natarajan, S., and Kanevsky, A. Fixed-priority scheduling of real-time systems using utilization bounds. *Journal of Systems and Software*, 33(1):57–63, April 1996.

[33] Vin, H., Goyal, P., and Goyal, A. A statistical admission control algorithm for multimedia servers. In *Proceedings of the 2nd ACM International Conference on Multimedia*, pages 33–40, New York, NY, USA, 1994. ACM.