Comparative Analysis of Real-Time Scheduling Algorithms on One Processor under Rate Monotonic

Omar U. Pereira Zapata, Pedro Mejía Alvarez, Luis E. Leyva del Foyo
CINVESTAV-IPN, Sección de Computación
Av. I.P.N. 2508, Zacatenco, México, D.F. 07300
{opereira, pmejia}@cs.cinvestav.mx, leyvadelfoyo@yahoo.com

May 13, 2005

Abstract

In this paper, a performance analysis is conducted for the best-known real-time schedulability tests executing under the Rate Monotonic Scheduling policy on one processor. The schedulability test verifies the fulfillment of the temporal constraints in a task set. We survey the inexact and exact schedulability conditions used in the scheduling of periodic and preemptable real-time tasks on uniprocessors using Rate Monotonic. Extensive simulation experiments are conducted to evaluate the inexact schedulability conditions and to compare their performance and computational complexity.

1 Introduction

Real-time systems are those in which its correct operation not only depends on the logical results, but also on the time at which these results are produced. These are high complexity systems that are executed in environments such as: military process control, robotics, avionic systems, distributed systems and multimedia.

Real-time systems use scheduling algorithms to decide an *order of execution* of the tasks and an *amount of time* assigned for 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. In order to verify the fulfillment of the temporal constraints, real-time systems use different exact or inexact *schedulability tests*. The schedulability test decides if a given task set can be scheduled such that no tasks in the set misses their deadlines. Exact schedulability tests usually have high time complexities and may not be adequate for online *admission control* where the system has a large number of tasks or a dynamic workload. In contrast, inexact schedulability tests provide low complexity sufficient schedulability tests.

The first schedulability test known was introduced by Liu and Layland with the Rate Monotonic Scheduling Algorithm [Liu, 1973] (RM). Liu and Layland introduced the concept of *achievable utilization factor* to provide a low complexity test for deciding the schedulability of independent periodic and preemptable task sets executing on one processor. The utilization factor of a task set is defined by $U = \sum_{i=1}^{n} C_i/T_i$, where C_i and T_i are the computation requirement and period of the task τ_i respectively and n is the number of tasks in the set.

The schedulability test introduced by Liu and Layland for RM states that a task set will not miss any deadline if it meets the following condition: $U \leq n(2^{1/n} - 1)$. Liu and Layland provided an schedulability tests that fails to identify many schedulable task sets when the system is heavily

overloaded. After the work of Liu and Layland, many researchers have introduced improvements on the schedulability condition for RM for one and multi processors. These improvements include the introduction of additional timing parameters in the schedulability tests and transformations on the task sets. It is a well known fact that when more timing parameters are introduced in the schedulability condition better performance can be achieved.

In this paper we survey the inexact and exact schedulability conditions used in the scheduling of periodic and preemptable real-time tasks on uniprocessors using Rate Monotonic. We analyze the best-known inexact schedulability conditions for one processor that exist in the literature, and conduct extensive simulation experiments to evaluate and compare their performance and computational complexity.

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 inexact schedulability conditions compared their performance only against the schedulability condition of Liu and Layland, but not against other scheduling algorithms.

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 scheme are introduced. In Section 5, extensions to the scheduling analysis are introduced, and in Section 6 extensive simulation experiments are conducted to test and compare the performance of the inexact schedulability conditions. Finally the conclusions appear in Section 7.

2 Real-Time Systems Scheduling

Real-time systems interact dynamically with a changing environment, on which several independent events must be controlled. Therefore a real-time system is composed of several concurrent activities (which normally are implemented as *tasks*).

The scheduling of non-real-time operating systems, supports the model of concurrent processes (tasks). This model exhibits a significative *non-determinism* because the correctness of the system does not depend on the precise order at which every task is executed (interleaved), and because its logical correctness is achieved using synchronization mechanisms to meet the ordering restrictions (i.e., mutual exclusion and conditional synchronization). In consequence, in traditional operating system the scheduler has the following general objectives: *optimal performance*, *optimal usage of the resources* and *fairness* in the resource assignment. Although the *logical correctness* of a concurrent program does not depend on the scheduling algorithm used, its *temporal correctness* does depend on the scheduling algorithm. Therefore, the scheduler of a real-time operating systems, must restrict the non-determinism associated with concurrent systems and provide the means to predict the worst-case temporal behavior of the system.

A real-time *scheduling algorithm* besides providing an *ordering policy* for the execution of the tasks (as in non real-time scheduling algorithms) also it provides a *schedulability test* to guarantee the temporal constraints of the application. The objective of any real-time scheduling policy must be to maximize the number of tasks that meet their deadlines. A *schedulability test*, is a mathematical test which verifies whether or not the set of task meets the temporal restrictions of the application. For a given scheduling algorithm, the task set is *schedulable* if it satisfies the schedulability test.

Real-time systems can be classified in two categories: *Hard* real-time systems and *Soft* real-time systems. In a hard real-time system, a delayed answer is an incorrect and unsuitable answer, whereas in a soft real-time system such delayed answer can be characterized giving a certain value of utility, depending on the lateness of this answer.

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 job(s) for a given task set may miss some of its 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 scheduling algorithms for hard real-time applications is their ability to find feasible schedules of the given application whenever such schedules exists. A hard real-time scheduling algorithm is *optimal* if for any feasible task set it always produces feasible schedules [Liu, 2000].

The scheduling algorithms can be classified in static and dynamic. In a static scheduling algorithm, all scheduling decisions are provided a priori. Given a set of timing constraints and a schedulability test, a table is constructed, using one of many possible techniques (e.g., using various search techniques), to identify the start and completion times of each task, such that no task misses their deadlines. This is a highly predictable approach, but it is static in the sense that when the characteristics of the task set change the system must be re-started and its scheduling table re-computed.

In a dynamic scheduling algorithm, the scheduling decision are executed at run-time based on task's priorities. The dynamic scheduling algorithms can be classified in algorithms with *fixed* priorities and algorithms with *variable* priorities. In the scheduling algorithms with fixed priorities, the priority of each task of the system remains static during the complete execution of the system, whereas in an algorithm with variable priorities the priority of a task is allowed to change at any moment.

The schedulability test in static scheduling algorithms can only be performed *off-line*, but in dynamic scheduling algorithms it can be performed off-line or *on-line*. In the off-line scheduling test, there are complete knowledge of the set of tasks executing in the system, as well as the restrictions imposed to each one of the tasks (deadlines, precedence restrictions, execution times), before the start of their execution. Therefore no new tasks are allowed to arrive in the system.

In contrast, in the on-line scheduling test, new arrivals are allowed and the tasks can change their timing constraints during their execution. In this test, the scheduler decides dynamically, by means of an admission control mechanism, if such new tasks can be accepted without interfering with the temporal constraints of the existing tasks in the system.

Liu and Layland [Liu, 1973] developed the first real-time scheduling algorithms for a single processor (Rate Monotonic and Earliest Deadline First), and developed their corresponding schedulability analysis for these algorithms. In [Liu, 1973] a *sufficient* condition for the scheduling of a set of periodic tasks with fixed priorities under RM is introduced, as well as a *sufficient and necessary* condition for the scheduling of a set of preemptable periodic tasks with dynamic priorities under EDF. RM assigns the highest priorities to tasks with smaller periods, and EDF assigns priorities to tasks considering the proximity of each instance of the tasks with its deadline (the task with closest deadline receives the highest priority). Liu and Layland [Liu, 1973] demonstrated that RM and EDF are optimal among all the fixed and dynamic priorities algorithms respectively.

2.1 Problem Definition

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

We assume that all tasks have a *phasing* relative to 0, I_i, with $0 \leq I_i < T_i$. This means that jobs corresponding to task τ_i are initiated at times $I_i + kTi$, $k \geq 0$. The job initiated at time $I_i + kT_i$ has $I_i + (k + l)T_i$ as its deadline, the initiation time of the next job. We label the tasks so that $T_1 \leq T_2, ..., \leq T_n$. We assume tasks are ready to run at their initiation times.

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 a *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 τ_i is not affected by arrival of any job of other task $\tau_{j \neq i}$ in the system.

A2 It is assumed that all the tasks in the system can be preempted at any time.

A3 The cost of the context switch of the tasks is considered negligible.

A4 The cost of the admission control mechanisms is considered null.

A5 No resources, other than the CPU is shared among tasks.

3 Schedulability Analysis

In the scheduling of any real-time application, the timing constraints of the application must be obtained along with a scheduling algorithm to execute the real-time tasks. Once all the timing constraints for all the tasks are defined, and a scheduling algorithm is chosen, the *schedulability analysis* verifies the fulfillment of the temporal constraints. The schedulability analysis developed can be classified in *exact tests* or *inexact tests*. The exact schedulability tests allows the system designer to decide whether a set of tasks is schedulable or not. That is, they provide a *necessary and sufficient test*.

A test is said to be sufficient in the sense that a task set is always schedulable if it satisfies the test. A test is said to be necessary if all schedulable task sets always satisfy the test.

The inexact schedulability tests provide a *sufficient* (but not necessary) schedulability condition.

Schedulability tests depend on the knowledge of several parameters of the task set and on for a given scheduling algorithm. Among all such sufficient tests, for a given set of task parameters, the largest test is called the *tight test*. In other words, the tight test is the best possible test that can be found using that task set knowledge.

Any test is tight only for a given task set parameters, so better tests can be obtained only providing additional parameters for the task set.

The schedulability tests are based on:

- *The utilization of the task set:*

  A utilization bound $\hat{U}$ of real-time task sets is the value such that any task set whose utilization factor is no larger than $\hat{U}$ is schedulable under any scheduling algorithm.

  The techniques based on the utilization, verify if the utilization of the set of tasks does not exceed a schedulability level (i.e., $U \leq \hat{U}$). The advantage of this test is its simplicity and low computational complexity. It provides a sufficient condition. The conditions based on the utilization found in the literature are: the Lui & Layland condition (L&L) developed for Rate Monotonic by Liu & Layland [Liu, 1973] and the condition Utilization Oriented (UO) introduced by Y. Oh et al. [Oh, 1995]. Some variants of the utilization tests, use additional information from the task set. In these tests, information of the period of the tasks is included in the analysis.

The most common schedulability conditions that use the period are: condition IP (Increasing Period) [Dhall, 1978], condition PO (Period Oriented) [Burchard, 1995], the RBOUND condition [Lauzac, 2003], the conditions based on *harmonic chains* [Kuo, 1991, Kuo, 2000, Han, 1997], and the algorithms of Chen, Mok & Kuo [Chen, 2003].

- *The response times:* [Joseph, 1986]. The objective of these tests is to determine the maximum response time of each task of the system, r_i. This test is solved using a recurrence equation. If for all the tasks in the system $r_i < D_i$, then the system is schedulable. Otherwise the system is not schedulable. The advantage of this test is that it is an exact test that obtains the schedulability of individual tasks. The theoretical computational complexity of this test is high.

- *The processor's demand of time* [Lehoczky, 1989]. These techniques are based on verifying that the total demand of the processor, by the task set in certain time intervals, does not exceed the maximum capacity of the processor. This technique is exact, but demands high computational complexity.

4 Schedulability Conditions for Fixed-Priority Scheduling on a Single Processor

In the following sections we will introduce the best-known schedulability conditions found in the literature, for Rate Monotonic on one processor.

4.1 Schedulability Condition L&L (Liu and Layland)

L&L is a schedulability condition for task sets scheduled under RM, that is based on the utilization of the processor [Liu, 1973]. In this condition, the utilization of the task set is compared with a utilization bound, which depends only on the number of tasks in the system. A task set will not miss any deadline if it meets the following condition:

$$U_n \leq n(2^{1/n} - 1) \tag{1}$$

This condition will be denoted as the Liu and Layland Condition (L&L), and it will be formalized in the following theorem.

Theorem 1 ***(Condition L&L)*** *[Liu, 1973]: If a set of tasks τ is schedulable under RM algorithm, then the minimum utilization achieved is $n(2^{1/n} - 1)$. When $n \to \infty$, then $n(2^{1/n} - 1) \to ln\ 2$.*

In this condition, it is important to note that no other information but the number of tasks is necessary. This condition is tight[Liu, 1973].

Condition L&L is inexact and its complexity is $O(n)$. Liu and Layland [Liu, 1973] proved that the worst case phasing occurs when $I_i = O$ for $1 \leq i \leq n$. This is called a *critical instant* in which all tasks are simultaneously instantiated. Using this concept, Liu and Layland also proved that the task set is schedulable (all deadlines of all jobs of every task are met) using the rate monotonic algorithm if the first job of each task can meet its deadline when it is initiated at a critical instant.

Figure 1 shows the utilization factor of the processor following the schedulability condition L&L. It can be observed that when the number of tasks tends to infinite, the *minimum achievable utilization* tends to $ln(2)$.

τ_i	τ_1	τ_2	τ_3	τ_4	τ_5
T_i	8	16	3	12	48
C_i	1	3	1	2	6
u_i	0.125	0.1875	0.3333	0.1666	0.1250
U	0.125	0.3125	0.6458	0.8124	0.9374

Table 1: Example Task Set

For the case of the Deadline Monotonic Algorithm (DM), in which the deadlines of the tasks are smaller or equal to the period, ($D_i \leq T_i$), Leung and Whitehead [Leung, 1982] generalized the results provided by Liu and Layland [Liu, 1973] and proved that the DM algorithm is optimal for the fixed priority scheduling scheme.

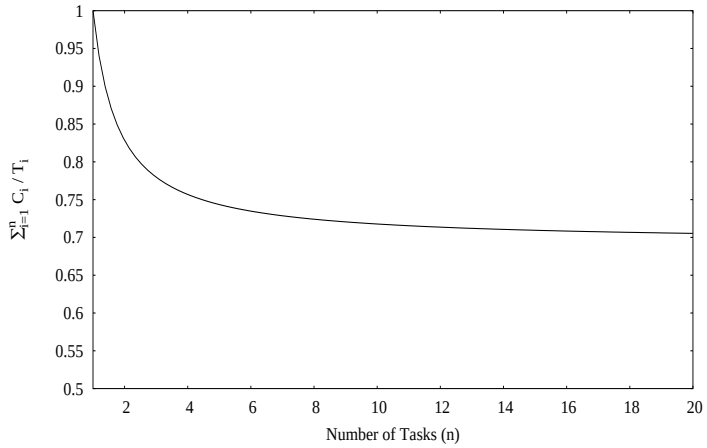

Figure 1: Performance of Condition L&L

Example 1.

In this example we will show the performance of the L&L schedulability condition.

Table 1 shows a task set τ with 5 tasks, including the timing constraints for each task, C_i, T_i, and u_i and $U = \sum_{\{j=1,\ldots,i\}} u_j$.

After applying condition L&L to the task set described in Table 1 it is observed that τ_1, τ_2 and τ_3 can be accepted in the system, since $u_1 + u_2 + u_3 = 0.125 + 0.1875 + 0.3333 = 0.6458 < 3(2^{1/3} - 1) = 0.7797$. However, adding task τ_4 or task τ_5, to the system will violate the L&L condition, $\sum_{i=1}^{4} u_i = 0.8124 > 4(2^{1/4} - 1) = 0.7568$, and $0.125 + 0.1875 + 0.333 + 0.1250 = 0.7708 > 0.7568$.

4.2 Schedulability Condition IP (Increasing Period)

Condition IP, introduced by Dhall and Liu [Dhall, 1978] was implemented on the multiprocessor algorithms Rate Monotonic Next Fit and Rate Monotonic First Fit.

Although condition L&L only depends on the number of tasks in the set, there are other conditions oriented to the periods or to the utilization of the tasks.

Theorem 2 ***(Condition IP)*** *[Dhall, 1978]: Let $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be a set of tasks with $T_1 \leq T_2 \leq \ldots \leq T_n$ and let*

$$U_{n-1} = \sum_{i=1}^{n-1} C_i \,/\, T_i \;\leq\; (n-1)(2^{1/(n-1)} \;-\; 1) \tag{2}$$

If the following condition is met,

$$u_n \;\leq\; 2\Big(1 + \;\frac{U_{n-1}}{(n-1)}\Big)^{-(n-1)} \;-\; 1 \tag{3}$$

then the set of tasks will have a feasible schedule under the RM algorithm. When $n \rightarrow \infty$*, the minimum utilization of task* τ_n *approaches to* $(2\, e^{-u} \;-\; 1)$*.*

Note that in this condition an ordering of the periods of the tasks is required. Because of this its complexity is $O(n\ log\ n)$.

As can be noted this condition is based on the utilization (and on the number of tasks already in the system) and it is inexact.

In Figure 2 the utilization of task τ_n is shown as a function of the $(n-1)$ tasks already accepted in the system. The different curves shown in the figure illustrate different values from the number of tasks (n). The area under the curve represents the *feasibility area*. For example, if in the uniprocessor system there is only one task with utilization of 40%, then condition IP will accept any other task with utilization of at most $(2(1+0.4)^{-1} - 1) \approx 0.4285 \approx 42\%$.

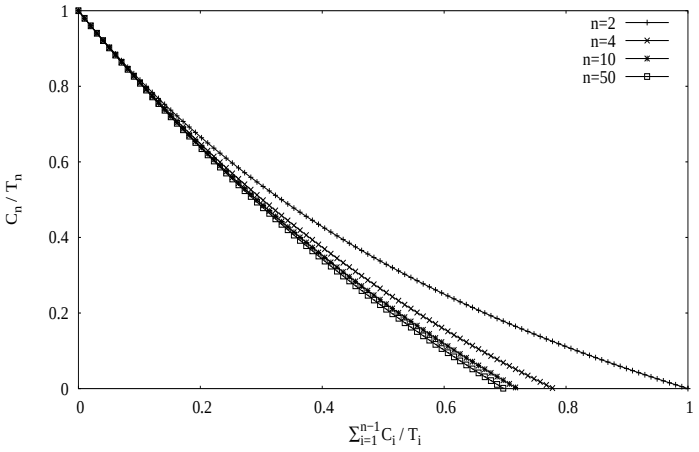

Figure 2: Performance of Condition IP.

Example 2.

In this example we will show the performance of the IP schedulability condition.

Before using condition IP for the task set described in Table 1, it is necessary to order the tasks according to an increasing order of their periods. After applying condition IP, it can be observed that tasks τ_3, τ_1 and τ_4 can be accepted to the system. Task τ_2 cannot be accepted because it violates condition IP, $u \approx 0.3333 + 0.125 + 0.166 = 0.6243$, $2(1 + 0.6243/3)^{-3} - 1 \approx 0.1342$, $u_2 = 0.1875 \;>$ 0.13428, but since $u_5 = 0.125 < 0.1342$, task τ_5 can be accepted.

4.3 Schedulability Condition PO (Period Oriented)

Condition PO, was introduced by Burchard et al. [Burchard, 1995] in the development of the RMST (Rate Monotonic Small Tasks) and RMGT (Rate Monotonic General Tasks) multiprocessor algorithms.

This condition considers that it is possible to increase the utilization of a task set if the periods of all tasks are close to each other.

Theorem 3 ***(Condition PO)*** *[Burchard, 1995]: Given a set of tasks $\tau = \{\tau_1, \ldots, \tau_n\}$, S_i and β are defined as follows,*

$$S_i = log_2\ T_i - \lfloor\ log_2\ T_i\ \rfloor \qquad i = 1, \ldots, n \tag{4}$$

and

$$\beta = max\ S_i - min\ S_i \qquad i = 1, \ldots, n \tag{5}$$

(a) if $\beta\ <\ (1\ -\ 1/n)$ and the total utilization satisfies that:

$$U\ \leq\ (n-1)(2^{\beta\ /\ (n\ -\ 1)}\ -\ 1)\ +\ 2^{1-\beta}\ -\ 1 \tag{6}$$

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

(b) if $\beta\ \geq\ (1\ -\ 1/n)$ and the total utilization satisfies that:

$$U\ \leq\ n(2^{1/n}\ -\ 1) \tag{7}$$

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

Condition (b) is similar to condition L&L [Liu, 1973], however, in general condition PO has better performance than L&L when Condition (a) is met.

Condition PO, in its simplest version, is defined in Corollary 1.

Corollary 1 ***(Condition PO)*** *[Burchard, 1995]: Given a set of tasks $\tau = \{\tau_1, \ldots, \tau_n\}$ and given β (defined as in Theorem 3), if the total utilization satisfies that:*

$$U = \sum_{i=1}^{n} C_i/T_i \leq max\{ln\ 2, 1 - \beta\ ln\ 2\} \tag{8}$$

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

In this condition it is necessary to know the periods of all tasks in the system. This condition is tight [Burchard, 1995] for the set of parameters used in the condition.

Figure 3, shows the performance of Condition PO. It shows that, depending on the number of tasks (n) and on the value of β, the task set can be feasibly scheduled if the utilization of the task set lies bellow the area of the curve. When the number of tasks is high and the value of $\beta = 1$ (meaning that the periods are not *harmonic*), then the minimal achievable utilization is approximately 69% (similar to the result provided by Liu and Layland in their L&L Condition [Liu, 1973]).

Example 3.

In this example we will show the performance of the PO schedulability condition.

The first step on applying condition PO to the task set described in Table 1 is to calculate the S_i values from Equation 4, and to order them in a nondecreasing fashion. S_i values computed are: $\{S_1 = 0, S_2 = 0, S_3 = 0.5849, S_4 = 0.5849, S_5 = 0.5849\}$. From this set it can be observed that tasks τ_1, τ_2 and τ_3 are schedulable, while task τ_4 violates the condition, since $\sum_{i=1}^{4} u_i = 0.8124\ >\ 3(2^{0.5849/3} - 1) + 2^{1-0.5849} - 1$.

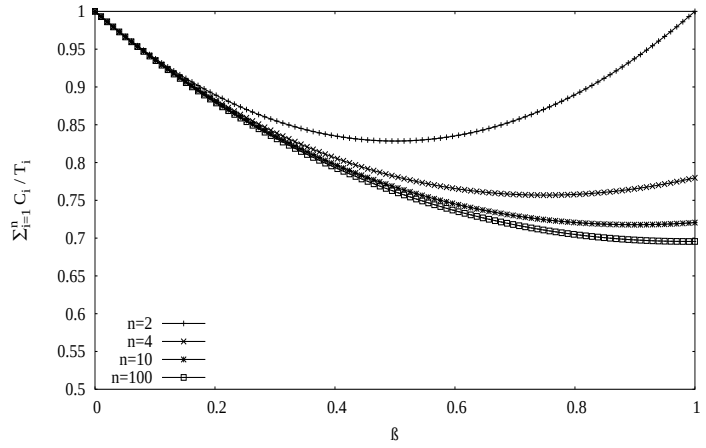

Figure 3: Performance of Condition PO.

4.4 Schedulability Condition UO (Utilization Oriented)

Y. Oh et al. [Oh, 1995] introduced a schedulability condition based on the utilization of the tasks, in the development of the RM-FFDU multiprocessor algorithm.

Theorem 4 ***(Condition UO)*** *[Oh, 1995]: Let* $\tau = \{\tau_1, \tau_2, \ldots, \tau_{n-1}\}$ *be a set of* $(n-1)$ *tasks feasibly scheduled under RM. A new task* τ_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.*

$$C_n/T_n \leq 2[\prod_{i=1}^{n-1}(1+u_i)]^{-1} - 1 \tag{9}$$

Figure 4, shows the utilization bound UO for different values of n, where the *x-axis* denotes the utilization of the $(n-1)$ tasks and the *y-axis* denotes the utilization of task τ_n. The area under the curve show the feasibility area.

Note that, in this condition, besides taking into consideration the number of tasks, the utilization of the tasks is also considered.

This condition if tight for the set of parameters included in the condition. The complexity of this condition is lineal, $O(n)$.

Bini et al. [Bini, 2001] introduced a schedulability test similar to that provided in Condition UO (which was named *Hyperbolic Bound*), using the following equation.

$$\prod_{i=1}^{n}(u_i+1) \;\leq\; 2 \tag{10}$$

It is possible to note that Equation 10, can be derived from Equation 9. In [Bini, 2001] the hyperbolic bound was extended to include resource sharing and aperiodic servers.

Example 4.

In this example we will show the performance of the UO schedulability condition. using the task set described in Table 1.

In Condition UO tasks τ_1, τ_2 and τ_3 from Table 1 are schedulable, but task τ_4 does not meet the condition, because $u_4 = 0.1666 \;>\; 2[\prod_{i=1}^{3}(1+u_i)]^{-1} - 1 = 0.1228$.

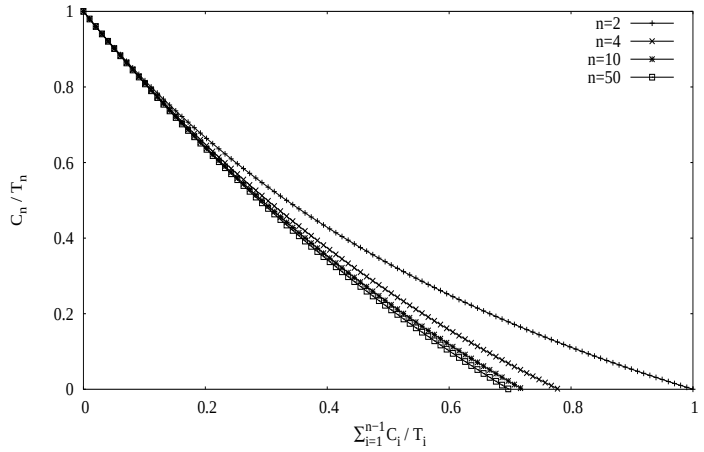

Figure 4: Performance of Condition UO.

4.5 Schedulability Condition RBOUND

Lauzac et al. [Lauzac, 2003] developed the schedulability condition RBOUND for the RM scheduling policy. This condition uses information of the periods of the task set to achieve a high utilization.

In order to apply condition RBOUND to a task set, first it is necessary to transform the original task set to an equivalent task set where the ratio between maximum and minimum periods is less than 2, $r = T_{max}/T_{min} \leq 2$. Then, the RBOUND condition is applied to the modified task set to verify its schedulability. As shown in Lemma 1, if the transformed task set is feasibly scheduled then the original task set is also feasible. Condition RBOUND is described in Theorem 5. The transformation procedure is made by the algorithm *ScaleTaskSet*, defined in Figure 5.

Lemma 1 *[Lauzac, 2003]: Let τ be a given periodic task set τ, and let τ' be the transformed task set after applying the ScaleTaskSet algorithm to τ. If τ' is schedulable on one processor under RM, then τ is also schedulable.*

ScaleTaskSet (In: τ, Out: τ')

```
1.  begin
2.        Sort the task set in τ by increasing period
3.        for (i = 1 to n − 1) do
4.                  T'_i = T_i 2^⌊ log Tn/Ti ⌋
5.                  C'_i = C_i 2^⌊ log Tn/Ti ⌋
6.        Sort the tasks in τ' by increasing period
7.        return (τ')
6.  end
```

Figure 5: Algorithm ScaleTaskSet

Theorem 5 ***(Condition RBOUND)*** *[Lauzac, 2003]: Consider a periodic task set τ, and let τ' be the transformed task set after executing the ScaleTaskSet on τ. If,*

$$\sum_{i=1}^{n} C_i/T_i \;\leq\; (n-1)(r^{1/(n-1)} - 1) \;+\; (2/r) \;-\; 1 \qquad (11)$$

where $r = T'_n/T'_1$, the task set τ can be feasibly scheduled on one processor under RM.

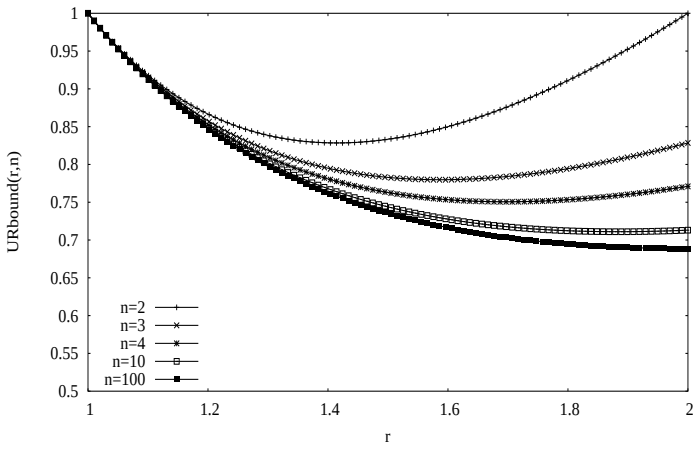

Figure 6: Performance of Schedulability Condition RBOUND.

Corollary 2 ***(Condition RBOUND)*** *[Lauzac, 2003]: When $n \rightarrow \infty$ the minimum achievable processor utilization $U_{RBound}(r)$ approaches to $(ln\ r) + (2/r) - 1$.*

Condition RBOUND is an inexact condition with complexity $O(n\ log\ n)$. Figure 6 shows the processor utilization U_{RBound} as a function of r. Note that for a given value of n, the minimum value of the curve yield the L&L utilization bound. This implies that RBOUND aways achieve a performance better or equal than condition L&L.

Algorithm RBOUND was extended for multiprocessors with algorithm RBOUND-MP[Lauzac, 2003], and for fault tolerance [Lauzac, 2003].

Example 5.

In this example we will show the performance of the RBOUND schedulability condition using the task set described in Table 1.

RBOUND [Lauzac, 2003] is more complex than previous conditions. First it is necessary to transform the task set from Table 1 using the $ScaleTaskSet$ procedure. This gives the following transformed task set: $\{\ \tau_1 = (T_1 = 32, C_1 = 4), \tau_2 = (T_2 = 32, C_2 = 6), \tau_3 = (T_3 = 48, C_3 = 16), \tau_4 = (T_4 = 48, C_4 = 8), \tau_5 = (T_5 = 48, C_5 = 6)\}$, where $T_i/T_j \leq 2$, for any $i \neq j$. Using the transformed task set, RBOUND finds τ_1, τ_2 and τ_3 schedulable, but task τ_4 violates the condition, since $\sum_{i=1}^{4} u_i = 0.8124 \ > \ 3(r^{1/3} - 1) + 2/r - 1$. Variable r, the ratio between the higher and smaller periods is $r = 48/32 = 1.5$.

4.6 Exact Schedulability Condition (Le)

Lehoczky et al. [Lehoczky, 1989] introduced a necessary and sufficient (exact) schedulability condition for task sets executing under RM.

Theorem 6 ***(Condition Le)*** *[Lehoczky, 1989]: Let $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be a task set with n tasks and $T_1 \leq T_2 \leq \ldots \leq T_n$. τ_i can be feasibly scheduled under RM if and only if,*

$$L_i = min\ _{\{t\ \in\ S_i\}}\ (W_i\ (t)\ /\ t) \leq\ 1 \tag{12}$$

where

$$S_i = \{\ k\ T_j \mid j = 1, \ldots, i;\ k = 1, \ldots,\ \lfloor\ T_i/T_j\ \rfloor\ \},$$
$$W_i\ (t) = \sum_{j=1}^{i}\ C_j\ \lceil\ t/T_j\ \rceil$$

The entire task set can be feasibly scheduled under RM if and only if

$$L = max_{\ \{\ 1 \leq i \leq n\ \}}\ L_i\ \leq\ 1 \tag{13}$$

The elements of S_i are know as the scheduling points for task τ_i [Lehoczky, 1989].

Theorem 6 gives a necessary and sufficient condition for the first job of each task to meet its deadline under the worst-case phasing (critical instant).

It can be observed that its computational complexity is pseudo-polynomial. Lehoczky [Lehoczky, 1990] extended this condition for the case when the deadlines of the tasks are less or equal than their periods, (i.e., $D_i \leq T_i$).

Example 6.

In this example we will show the performance of the *Le* schedulability condition using the task set described in Table 1.

The first step is to order the task set according to their periods. $\tau_1(T_1 = 3, C_1 = 1, u_1 = 0.3333)$, $\tau_2(T_2 = 8, C_2 = 1, u_2 = 0.125)$, $\tau_3(T_3 = 12, C_3 = 2, u_3 = 0.1666)$, $\tau_4(T_4 = 16, C_4 = 3, u_4 = 0.1875)$, $\tau_5(T_5 = 48, C_5 = 6, u_5 = 0.125)$.

It can be observed that tasks τ_1, τ_2, and τ_3 are schedulable according to $L\&L$ condition. That is, $u_1 + u_2 + u_3 \leq 3(2^{1/3} - 1)$, $0.6249 \leq 0.7797$.

To verify the schedulability of task τ_4 the time interval $[0, 16]$ has to be analyzed, assuming that time $t = 0$ is a critical instant.

The scheduling points of task τ_4 are the following: $S_4 = 1 * T_1, 2 * T_1, 3 * T_1, 4 * T_1, 5 * T_1, 1 * T_2, 2 * T_2, 1 * T_3, 1$ $3, 6, 8, 9, 12, 15, 16$. Task τ_4 is schedulable if $L_4 = min_{t\ \in\ S_i} W_4(t) \leq t$ holds.

t	$W_4(t)$
3	1*1 + 1*1 + 2*1 + 3*1 = 7
6	1*2 + 1*1 + 2*1 + 3*1 = 8
8	1*3 + 1*1 + 2*1 + 3*1 = 9
9	1*4 + 1*2 + 2*1 + 3*1 = 11
12	1*4 + 1*2 + 2*1 + 3*1 = 11

Table 2: $W_4(t)$

From Table 2 it can be concluded that task τ_4 is schedulable.

Following a similar procedure for task τ_5 we conclude that τ_5 is schedulable.

4.7 Response Time Analysis

Joseph and Pandya introduced an exact schedulability condition in [Joseph, 1986] for fixed priority scheduling. In this test, the response time of each task in set, r_i is obtained, and if $r_i \leq D_i$, then task τ_i meets its deadline.

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

$$r_1 = C_1$$

In order to obtain the response time of the remaining tasks, it is necessary to compute the *time interference* produced by high priority tasks on the low priority tasks. When this time interference is added to the response time of r_i we obtain,

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

where $hp(i)$ is the task set with higher priority than task τ_i. Since r_i appears at both sides of the equation, a possible solution is obtained by the following iterative process [Joseph, 1986].

Let w_i^n be the response time of task τ_i on iteration n.

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j \tag{14}$$

Iterations described on Equation (14) can start considering $W_i^0 = \sum_{k=1}^{i} C_k$. It is easy to note that $w_i^{n+1} \geq w_i^n$. If $w_i^n > D_i$ then task τ_i will miss its deadline. However, if $w_i^n = w_i^{n+1}$, the iterative process will finish and $r_i = w_i^n$.

The response time analysis has evolved to include offsets, blocking time, fault tolerance and release jitter [Audsley, 1993].

Example 7.

In this example we will show the performance of the *Le* schedulability condition using the task set described in Table 1. The first step is to order the task set according to their periods. Equation 14 is used to verify the schedulability of the task set.

Using the iterative equation for task τ_5 we obtain the following.

$w_5^0 = 13$
$w_5^1 = 6 + \lceil 13/3 \rceil * 1 + \lceil 13/8 \rceil * 1 + \lceil 13/12 \rceil * 2 + \lceil 13/16 \rceil * 3 = 20$
$w_5^2 = 6 + \lceil 20/3 \rceil * 1 + \lceil 20/8 \rceil * 1 + \lceil 20/12 \rceil * 2 + \lceil 20/16 \rceil * 3 = 26$
$w_5^3 = 6 + \lceil 26/3 \rceil * 1 + \lceil 26/8 \rceil * 1 + \lceil 26/12 \rceil * 2 + \lceil 26/16 \rceil * 3 = 31$
$w_5^4 = 6 + \lceil 31/3 \rceil * 1 + \lceil 31/8 \rceil * 1 + \lceil 31/12 \rceil * 2 + \lceil 31/16 \rceil * 3 = 33$
$w_5^5 = 6 + \lceil 33/3 \rceil * 1 + \lceil 33/8 \rceil * 1 + \lceil 33/12 \rceil * 2 + \lceil 33/16 \rceil * 3 = 37$
$w_5^6 = 6 + \lceil 37/3 \rceil * 1 + \lceil 37/8 \rceil * 1 + \lceil 37/12 \rceil * 2 + \lceil 37/16 \rceil * 3 = 41$
$w_5^7 = 6 + \lceil 41/3 \rceil * 1 + \lceil 41/8 \rceil * 1 + \lceil 41/12 \rceil * 2 + \lceil 41/16 \rceil * 3 = 43$
$w_5^8 = 6 + \lceil 43/3 \rceil * 1 + \lceil 43/8 \rceil * 1 + \lceil 43/12 \rceil * 2 + \lceil 43/16 \rceil * 3 = 44$
$w_5^9 = 6 + \lceil 44/3 \rceil * 1 + \lceil 44/8 \rceil * 1 + \lceil 44/12 \rceil * 2 + \lceil 44/16 \rceil * 3 = 44$

Since the lowest priority task is schedulable ($w_5^9 = w_5^8 \leq D_5$) we conclude that tasks τ_1, τ_2, τ_3, and task τ_4 are also schedulable.

4.8 Conditions based on Harmonic Chains

In recent years, some researchers developed polynomial-time schedulability tests that improve the WC schedulability condition [Liu, 1973]. These new tests transform the task sets into a equivalent task sets were their periods tend to be *harmonic*.

A harmonic chain is a list of numbers (periods) in which each number divides every number after it [Chen, 2003].

4.8.1 HC Condition

Kuo and Mok [Kuo, 1991] developed the condition HC (*Harmonic Chain*), in which a periodic task set τ will find a feasible schedule if its utilization factor is no larger than $K(2^{1/K} - 1)$, where K is the size of a *harmonic base* of τ.

Definition 1 ***(Harmonic Base of τ)*** *[Kuo, 1991]: Let P be the set of periods (positive numbers) of a set τ of periodic tasks. A subset H of P is said to be a* harmonic base *of the task set τ if there is a* partition*, say Γ, of P into $|H|$ subsets, such that:*

1. *Each member of H is the smallest element in exactly one member of the partition Γ, and*
2. *If x and y are two elements in the same member of the partition Γ, then either x divides y or y divides x.*

Each subset in the partition Γ is called a *harmonic chain* [Kuo, 1991].

Theorem 7 ***(Condition HC)*** *[Kuo, 1991]: Let τ be a set of periodic tasks and let K be the size of a harmonic base of τ. If the utilization factor of τ is no larger than $K(2^{\frac{1}{K}} - 1)$, then τ is schedulable by a preemptive fixed priority scheduler.*

The following example will be used to clarify the concept of harmonic base.

Let P be the set of periods $P = \{3, 5, 15, 20, 60\}$ for the task set $\tau = \{\tau_1 = (T_1 = 3, C_1 = 1), \tau_2 = (T_2 = 5, C_2 = 1), \tau_3 = (T_3 = 15, C_3 = 2), \tau_4 = (T_4 = 20, C_4 = 3), \tau_5 = (T_5 = 60, C_5 = 8)\}$. Subset $H = \{3, 5\}$ is a harmonic base of P, because there exists a partition Γ in $|H|$ subsets, $\Gamma = \{\{3, 15\}, \{5, 20, 60\}\}$, such that (1) each member of H is the smallest single element of partition Γ, and (2) for each pair of elements within partition Γ, one of them divides the other element.

The problem of computing the harmonic base of a periodic task set can be solved by a polynomial time algorithm. It can be observed that when the size of the harmonic base is small, the utilization factor is large. For instance, if $K = 1$, then the CPU can be 100% utilized. Note that condition HC is similar to the condition L&L when all periods of the tasks are *relative primes* (two numbers a and b are relative primes, if they are non zeros and $MCD(a, b) = 1$).

Example 6.

In this example we will show the performance of the HC schedulability condition using the task set described in Table 1.

After applying condition HC[Kuo, 1991], we obtained that tasks τ_1, τ_2, τ_3 and τ_4 are schedulables, since their total utilization is no larger than $K(2^{1/K} - 1)$ where K is the size of the harmonic base of τ. From the example in Table 1 we observe that the size of the harmonic base of τ is 2. That is, $\sum_{i=1}^{4} u_i = 0.8124 \leq 2(2^{1/2} - 1)$. Note that the introduction of task τ_5 violates condition HC because the utilization of the task set increases beyond $2(2^{1/2} - 1)$.

4.8.2 ROOT Condition

Kuo et al. [Kuo, 2000] developed the condition ROOT, and demonstrated that a task set can be feasibly scheduled as long as the utilization of the task set is no larger than $R(2^{1/R} - 1)$, where R is the number of roots in the task set.

Definition 2 ***Root*** *[Kuo, 2000]: Let $\tau = \{\tau_1, \ldots, \tau_i, \ldots, \tau_n\}$ be a periodic task set. Task τ_i is a root in τ if there does not exists any task period in τ which is larger than and can be divided by the period of task τ_i.*

Theorem 8 ***(Condition ROOT)*** *[Kuo, 2000]: Suppose that the task set $\{\tau_1, \tau_2, \ldots, \tau_{i-1}\}$ is schedulable. Let R be the number of roots in the task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_{i-1}, \tau_i\}$. If the total utilization factor of τ is no larger than $R(2^{1/R} - 1)$, then τ is schedulable.*

Since the number of roots in τ is much less than the number of tasks (and also less than the size of its harmonic base), then it is expected that condition ROOT improves conditions L&L and HC. This can be observed in Corollaries 3, 4 and 5.

Corollary 3 *[Kuo, 2000]: Let τ be a set of periodic tasks. If τ is guaranteed to be schedulable according to Condition L&L, then τ is guaranteed to be schedulable according to Condition ROOT.*

Corollary 4 *[Kuo, 2000]: Let τ be a set of periodic tasks. If τ is guaranteed to be schedulable according to Condition HC, then τ is guaranteed to be schedulable according to Condition ROOT.*

Corollary 5 *[Kuo, 2000]: There exists a task set that is guaranteed to be schedulable according to Condition ROOT, but not according to Conditions L&L or HC.*

An important feature of Condition ROOT is that it was implemented incrementally [Kuo, 2000] for on-line admission control. The incremental algorithm of ROOT does not need to recompute the response times of all tasks, but only the newly arrived task.

Example 7.

In this example we will show the performance of the ROOT schedulability condition. using the task set described in Table 1.

Using condition ROOT for our example task set, it can be noted that tasks τ_2 and τ_4 are the *roots* of tasks τ_1, τ_2, τ_3 and τ_4. So, the schedulability condition ROOT is met for tasks τ_1, τ_2, τ_3 and τ_4, since $\sum_{i=1}^{4} u_i = 0.8124 \leq 2(2^{1/2} - 1)$. While considering task τ_5 it can be noted that this task is the root of tasks $\tau_1, \tau_2, \tau_3, \tau_4$ and τ_5. Therefore, task τ_5 is also schedulable since $\sum_{i=1}^{5} u_i = 0.9374 \leq 1(2^1 - 1)$.

4.8.3 H&T Conditions

Han and Tyan [Han, 1997], instead of searching for harmonic chains in the task periods, introduced a polynomial-time schedulability test which transform the task periods into a special pattern, where all periods belong to a single harmonic chain. That is, in this condition, given a task set τ, a transformed task set τ' is computed. If the total utilization of this transformed task set τ', is less than 1 and condition 1 is satisfied, then the task set τ is schedulable under RM.

Condition 1 *[Han, 1997]: $T'_i \leq T_i$ for all $i = 1, 2, \ldots, n$, and T'_i evenly divides T'_{i+1}, denoted as $T'_i | T'_j$, (thus, $T'_i \leq T'_{i+1}$) for all $i = 1, 2, \ldots, n-1$.*

The first algorithm proposed to find a task set τ' that satisfies condition 1 is called *algorithm SR* (*Specialization Operation*)[Han, 1997]. In this algorithm, each period T_i of task set τ 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 $(T_1/2, T_1]$. It is easy to see that $T'_i \leq T_i$, for all i and $T_i | T'_i$ for all $i < j$. Since $T'_i \leq T_i$ for all i, $U(\tau') \geq U(\tau)$. The problem is how to find the *best* r value, such that the total utilization increase $\Delta = U(\tau') - U(\tau)$ is minimized. Algorithm SR, illustrated in Figure 7, finds the best value for r, and then derive the new periods T'_i, for all i, using the best r.

Define $l_i = T_i / 2^{\lceil log(T_i/T_1) \rceil}$, for $1 \leq i \leq n$, where $(T_1/2 < l_i \leq T_1)$. Let $k_1 < k_2 < \ldots < k_u, u \leq n$, be the sorted sequence of l_i's with duplicates removed. Since $l_1 = T_1$, we know that $k_u = T_1$. We call $\{k_1, k_2, \ldots, k_u\}$ the *special base* of τ. The value of r that minimizes the total utilization increase

Δ, denoted by r^*, can always be found in the special base. Let $\phi_\tau(r)$ be the total utilization of the task set τ' with its periods $\{T'_1, T'_2, \ldots, T'_n\}$ specialized from $\{T_1, T_2, \ldots, T_n\}$ with respect to r, and let $\phi^*_\tau = \phi_\tau(r^*) = min_{\{T_1/2<r\leq T_1\}}\phi_\tau(r)$. We can compute $\phi_\tau(k_v)$ for all k_v in the special base of τ, select the one that result in the minimum value of $\phi_\tau(k_v)$, and use that k_v for r in the specialization operation.

In algorithm SR, π_r is called *r-based subset* of τ defined as: $\pi_r = \{\tau_i \in \tau | T_i = r * 2^j$, for some integer $j \geq 0\}$. The main reason to define the r-based subsets of τ is that for every task $T_i \in \pi_{k_v}$, $T_i = l_i \cdot 2^{\lceil log(T_i/T_1)\rceil} = k_v \cdot 2^{\lfloor log(T_i/k_v)\rfloor}$ (because $l_i = k_v$ and $\log(T_i/k_v) = \lfloor log(T_i/k_v)\rfloor = \lceil log(T_i/T_1)\rceil$ is an integer); as a result, if $r = k_v$, T_i will be specialized to itself and hence, the utilization of τ_i will not increase after the specialization operation. This also gives the intuition that why only the numbers in the special base of τ need to be considered in finding r^*.

Input: $\tau = \{\tau_i = (C_i, T_i) \mid 1 \leq i \leq n\}$, where τ is a periodic task set and $T_i \leq T_j$, for all $i < j$.
Output: A task set τ' and $\phi_\tau(r^*)$.

1. **begin**
2. **for** $(i = 1$ **to** $n)$ **do** $l_i = T_i/2^{\lceil log(T_i/T_1)\rceil}$
3. sort $(l_1, l_2, \ldots, l_n)$ into nondecreasing order and remove duplicates.
   -- Let $(k_1, k_2, \ldots, k_u)$ be the resulting sequence;
4. **for** $(i = 1$ **to** $n)$ **do** put τ_i into subset π_{l_i}
5. **for** $(v = 1$ **to** $u)$ **do** $U(\pi_{k_v}) = \sum_{\tau_i \in \pi_{k_v}} C_i/T_i$
6. compute $\phi_\tau(k_u) = \phi_\tau(T_1)$
7. **for** $(v = u - 1$ **downto** 1**)** **do**
   $\phi_\tau(k_v) = \frac{k_{v+1}}{k_v}\phi_\tau(k_{v+1}) - U(\pi_{k_v})$
8. Find r^* such that $\phi_\tau(r^*) = min_{r\in\{k_1,k_2,\ldots,k_u\}}\ \phi_\tau(r)$
9. **for** $(i = 1$ **to** $n)$ **do** $T'_i = r^* \cdot 2^{\lfloor log(T_i/r^*)\rfloor}$
10. **return** $\phi_\tau(r^*)$ and (τ')
11. **end**

Figure 7: SR Algorithm

Input: A task set τ with n tasks, with its periods ordered increasingly.
Output: A task set τ'.

1. **begin**
2. minf = -1; minutilization = ∞
3. **for** $f = 1$ **to** n **do**
4. $Z_f = T_f$.
5. **for** $(i = f + 1$ **to** $n)$ **do** $Z_i = Z_{i-1} * \lfloor T_i/Z_{i-1}\rfloor$.
6. **for** $(i = f - 1$ **downto** 1**)** **do** $Z_i = \frac{Z_{i+1}}{\lceil Z_{i+1}/T_i\rceil}$.
7. utilization = $\sum_{i=1}^{n} C_i/Z_i$.
8. **if** (utilization < minutilization) **then**
9. minutilization = utilization.
10. minf = f.
11. **for** $(i = 1$ **to** $n)$ **do** $T'_i = Z_i$.
12. **endif**
13. **return** (τ')
14. **end**.

Figure 8: DCT Algorithm

Example 8.

In this example we will show the performance of the SR algorithm using the task set described in Table 1. Following agorithm SR, the value of r denoted as r^* which minimizes the total utilization increase can always be found in the *special base* of the task set.

To obtain the special base it is necessary to execute the following procedure:

- order the task set;
- compute the values $l_i = T_i/2^{\lceil log(t_i/T1)\rceil}$;
- $l_1 = 3, l_2 = 2, l_3 = 3, l_4 = 2, l_5 = 3$;
- removing duplicates we obtain values 2 and 3.

Once the r values are obtained, the next step is to transform the period set using $T_i^{'} = r \ 2^{\lfloor (T_i/r)\rfloor}$. Therefore, for $r = 2$ we obtain:

$\tau_1(2,1)$ $u_1 = 0.5$
$\tau_2(8,1)$ $u_2 = 0.125$
$\tau_3(8,2)$ $u_3 = 0.25$
$\tau_4(16,3)$ $u_4 = 0.1875$
$\tau_5(32,6)$ $u_5 = 0.1875$
Total utilization = 1.25.

For $r = 3$ we obtain:

$\tau_1(3,1)$ $u_1 = 0.3333$
$\tau_2(6,1)$ $u_2 = 0.1666$
$\tau_3(12,2)$ $u_3 = 0.1666$
$\tau_4(12,3)$ $u_4 = 0.25$
$\tau_5(48,6)$ $u_5 = 0.125$
Total utilization= 1.0415.

From this procedure it can be observed that the value of r that minimizes the total utilizatiton increase is $r^* = 3$. However since the resulting utilization is greater than 1, the task set can not be feasibly scheduled using algorithm SR. In other words, SR cannot find a set of periods able to form a single fundamental frequency such that the task set utilization with this new periods be less or equal than 1.

The second algorithm proposed by Han and Tyan in [Han, 1997] is called *algorithm DCT*. The idea behind algorithm DCT is the following. For each $f, 1 \leq f \leq n$, T_f' is set to $T_f' = T_f$, and then 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,

$$T_i' = T_{i-1}' \cdot \lfloor T_i/T_{i-1}' \rfloor, for\ i = f+1, f+2, \ldots, n. \tag{15}$$

Similarly, T_i, for each $i < f$, is recursively transformed to the largest divisor of T_{i+1}' that is less than or equal to T_i. That is,

$$T_i' = \frac{T_{i+1}'}{\lceil T_{i+1}'/T_i \rceil}, for\ i = f-1, f-2, \ldots, 1. \tag{16}$$

The value of f that result 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. Given a periodic task set τ, we can use the SR or DCT algorithms to derive a task set which satisfy Condition 1 and if the total utilization of the derived

task set is less than or equal to 1, it means that the original task set τ is schedulable by RM. The computational complexity of Algorithms SR and DCT is $O(n\ log\ n)$ and $O(n^2)$ respectively. The main result, provided by Han and Tyan is given in the following Theorem.

Theorem 9 *[Han, 1997]: Given a task set τ, if there exists a transformed task set τ' which satisfies condition 1 and $U(\tau') = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$, then τ is schedulable under RM.*

Example 9.

In this example we will show the performance of the DCT algorithm using the task set described in Table 1. The approach followed by algorithm DCT is to recursively transform the periods of all tasks such that they form a single harmonic chain.

Following algorithm DCT described in Figure 8 we obtain:

For task τ_1:
$\tau_1(3,1)$ $u_1 = 0.3333$
$\tau_2(6,1)$ $u_2 = 0.1666$
$\tau_3(12,2)$ $u_3 = 0.1666$
$\tau_4(12,3)$ $u_4 = 0.25$
$\tau_5(48,6)$ $u_5 = 0.125$
The total utilization from the resulting task is 1.0415.

For task τ_2 we obtain:
$\tau_1(2.66,1)$ $u_1 = 0.375$
$\tau_2(8,1)$ $u_2 = 0.125$
$\tau_3(8,2)$ $u_3 = 0.25$
$\tau_4(16,3)$ $u_4 = 0.1875$
$\tau_5(48,6)$ $u_5 = 0.125$
The total utilization from the resulting task set is 1.0625.

For task τ_3 we obtain:
$\tau_1(3,1)u_1 = 0.3333$
$\tau_2(6,1)u_2 = 0.1666$
$\tau_3(12,2)u_3 = 0.1666$
$\tau_4(12,3)u_4 = 0.25$
$\tau_5(48,6)u_5 = 0.125$

The total utilization for the resulting tast set is 1.0415. Note that this transformation of periods gives a similar result than the one applied to task τ_1.

Applying the same transformation for tasks τ_4 and τ_5 we obtained similar results to those of tasks τ_2. From the prevous results the minimum utilization of 1.0415 which makes the resulting task set unschedulable using the DCT algorithm.

4.8.4 Chen, Mok & Kuo Algorithms

Chen, Mok and Kuo [Chen, 2003] developed three algorithms with polynomial-time complexity, that yield better bounds than the Liu and Layland Bound.

On algorithm 1 (illustrated in Figure 9), a task set τ is transformed into another task set, in which the ratio of any T_i and T_j (for all $i \neq j$), is no larger than 2. An utilization bound is developed based on Theorem 10. If the utilization of τ is less than the resultant utilization, U_{bound} (after applying algorithm 1), then the task set is schedulable under RM.

Theorem 10 *[Chen, 2003]: Let $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be a set of periodic tasks. Let $\vec{T}$ be the array of periods for the task set. If $T_1 < T_2 < \ldots < T_n < 2T_1$, then the exact utilization bound U_{bound} for the task set is obtained when,*

$$C_i = T_{i+1} - T_i,\ 1 \leq i < n \tag{17}$$

$$C_n = 2T_1 - T_n \tag{18}$$

$$U_{bound} = \sum_{i=1}^{n-1} \frac{T_{i+1} - T_i}{T_i} + \frac{2T_1 - T_n}{T_n} \tag{19}$$

Input: `Array Period[n] in nondecreasing order`
Output: `Utilization Bound` U_{bound}.

1. **begin**
2. U_{bound} `= 1;`
3. **for** $(i = 2$ **to** $n)$ **do**
4. **for** $(j = 1$ **to** $i)$ **do**
5. `NewPeriod[j] = Period[j] x` $\lfloor \frac{Period[i]}{Period[j]} \rfloor$;
6. `U =` $\sum_{j=1}^{i-1} \frac{NewPeriod[j+1] - NewPeriod[j]}{NewPeriod[j]} + \frac{2NewPeriod[1] - NewPeriod[i]}{NewPeriod[i]}$;
7. **if** $(U_{bound} > U)$ **then**
8. $U_{bound} = U$;
9. **return**(U_{bound})
10. **end**.

Figure 9: Algorithm 1

The second algorithm (illustrated in Figure 10), introduced by Chen, Mok and Kuo [Chen, 2003] developed an strategy to compute with higher efficiency the size of the *harmonic base* from a set of tasks, than that introduced in condition HC [Kuo, 1991]. Algorithm 2 is based on Lemma 2.

Input: `The array Period[n] in nondecreasing order`
Output: `Utilization bound` U_{bound}.

1. **begin**
2. **for** $i = 1$ **to** $n - 1$ **do**
3. `TasksAttr[i] =` ∞
4. **for** $(j = n$ **downto** $i + 1)$ **do**
5. **if** $((Period[j]$ **mod** $Period[i]) = 0)$ **then**
6. `TasksAttr[i] = Period[j];`
7. `Tasksattr[n] =` ∞;
8. $k = 0$;
9. **for** $(i = 1$ **to** $n)$ **do**
10. `j=0;`
11. **for** $(m = 1$ **to** $i)$ **do**
12. **if** `(TasksAttr[m]` $\leq$ `Period[i])` **then** $j = j + 1$;
13. **if** $(k < i - j)$ **then** $k = i - j$;
14. **return**$(U_{bound} = k(2^{1/k} - 1))$
10. **end**.

Figure 10: Algorithm 2

Input: The array Period[n] in a nondecreasing order
Output: Utilization Bound U_{bound}.

```
1.  begin
2.   U_bound = 1;
3.   for (i = 1 to n) do
4.       Reduced-period1 [] = Array P_reduced
5.       for (j=1 to i) do
6.           Reduced − period2[j] = Reduced − period1[j] × ⌊Reduced−period1[i] / Reduced−period1[j]⌋
7.       U = Utilization factor calculated from Reduced-period2
         using Theorem 10.
8.       if (U_bound > U) then
9.           U_bound = U;
10.  return(U_bound)
11.  end.
```

Figure 11: Algorithm 3

Definition 3 *For a given task set* $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ *with* $T_1 < T_2 < \ldots < T_n$, *let:*

$$T_n = t_i T_i + r_i,\ 0 \leq r_i < T_i \tag{20}$$

$$C_i = \frac{T_i - r_i}{T_i} \tag{21}$$

α_{ij} = *Number of instances of* τ_j *between* $t_i T_i$ *and* T_n *when* $t_i T_i \leq t_j T_j$.

Lemma 2 *[Chen, 2003]: If* T_i *divides* T_j, $1 \leq i < j \leq n$, *then the exact utilization bound* U_{bound} *from the period array* $\vec{T}$ *equals the exact utilization bound of* $\vec{T}' = [T_1, T_2, \ldots, T_{i-1}, T_{i+1}, \ldots, T_n]$.

Lemma 3 *[Chen, 2003]: Consider* T_j *and* T_k *in* $\vec{T}$, $t_n = t_j T_j + r_j = t_k T_k + r_k$. *If* $t_j T_j \leq t_k T_k$ *and* $C_k \leq \alpha_{jk} C_j$, *then the exact utilization bound* U_{bound} *from the period array* $\vec{T}$ *is equal to the exact utilization bound* $\vec{T}' = [T_1, T_2, \ldots, T_{k-1}, T_{k+1}, \ldots, T_n]$.

Definition 4 *(*$\vec{T}_{reduced}$ *array) [Chen, 2003]: Given an array of periods* $\vec{T}$, *assume all periods from this array are eliminated according to Lemmas 2 and 3. Let us denote the resultant period array:* $\vec{T}_{reduced}$ *the reduced array to* $\vec{T}$.

Algorithm 3 proposed in [Chen, 2003], is an improvement of algorithm 2, where the input data used is $\vec{T}_{reduced}$, instead of the $\vec{T}$ array. In algorithm 3, Theorem 10 is applied to obtain the utilization bound.

5 Extensions to Schedulability Conditions

The scheduling analysis described in previous sections has been extended to solve diverse problems related to synchronization [Sha, 1990], aperiodic tasks [Sprunt, 1989], resource sharing [Rajkumar, 1991, Lipari, 2000], fault-tolerance [Lauzac: 2000, Burns, 1996], non-preemptive tasks [Liu, 2000], and overload handling [Buttazzo, 1998].

Other studies developed schedulability conditions for aperiodic scheduling [Abdelzaher, 2004], precedence constrained scheduling [Liu, 2002], multiple processor scheduling [Andersson, 2001, Burchard, 1995, Lopez, 2003], and QoS scheduling [Lee, 2004].

Algorithm	Utilization Bound	Complexity	$lim_{n\to\infty}$	Restrictions	Remarks
L&L	$U \leq n(2^{1/n} - 1)$	O(n)	ln 2	- n must be known	- Sufficient condition - Tigth condition
IP	$u_n \leq 2(1 + \frac{U_{n-1}}{n-1})^{-(n-1)} - 1$	O(n log n)	$2e^{-u} - 1$	- n and T_i must be known - $T_1 \leq T_2 \leq \ldots \leq T_n$ - $U_{n-1} \leq (n-1)(2^{1/(n-1)} - 1)$ where $U_{n-1} = \sum_{i=1}^{n-1} u_i$	- Sufficient condition
PO	$U \leq (n-1)(2^{\beta/(n-1)} - 1) + 2^{1-\beta} - 1$	O(n log n)	$1 - \beta\ ln\ 2$	- n and T_i - $S_1 \leq S_2 \leq \ldots \leq S_n$ where $S_i = log\ T_i - \lfloor log\ T_i \rfloor$ - $\beta < 1/(1-n)$ where $\beta = max\ S_i - min\ S_i$	- Sufficient condition - Tigth condition
RBOUND	$U \leq (n-1)(r^{1/(n-1)} - 1) + \frac{2}{r} - 1$	O(n log n)	ln $r + \frac{2}{r} - 1$	- n and T_i must be known - $r = \frac{max\ T_i}{min\ T_i} \leq 2$ - $T_1 \leq T_2 \leq \ldots \leq T_n$	- Sufficient condition - $r \leq 2$ r = ratio of any two periods
UO, HB	$\prod_{i=1}^{n}(1 + u_i) \leq 2$	O(n)	N/A	- n and u_i must be known	- Sufficient condition - Condition is tight
HC	$U \leq k(2^{1/k} - 1))$	O($n^{5/2}$)	ln 2; n=k	- k must be obtained - k = number of harmonic chains	- Sufficient condition
ROOT	$U \leq r(2^{1/r} - 1))$	O(n^2)	$ln2; n = r$	- r must be computed r = number of roots	- Sufficient condition
IFF	$L = max_{\{1\leq i\leq n\}} L_i \leq 1$ where: $L_i = min_{\{t\in S_i\}} W_i(t)/t \leq 1$ $S_i = \{kT_j \mid j = 1, \ldots, i;$ $k = 1, \ldots, \lfloor T_i/T_j \rfloor\}$ $W_i(t) = \sum_{j=1}^{i} C_j \lceil t/T_j \rceil$	pseudo-polinomial	N/A	- n, C_i and T_i must be known - $T_1 \leq T_2 \leq \ldots \leq T_n$	- Necessary and sufficient condition
SR	N/A	O(n log n)	N/A	- n and T_i must be known - $T_1 \leq T_2 \leq \ldots \leq T_n$	- Transform the set of periods into a single harmonic chain
DCT	N/A	O(n^2)	N/A	- n and T_i must be known - $T_1 \leq T_2 \leq \ldots \leq T_n$	- Transform the set of periods into a single harmonic chain
Algorithm 1	N/A	O(n^2)	N/A	- n and T_i must be known - $T_1 \leq T_2 \leq \ldots \leq T_n$	- Algorithm 1 finds an utilization bound
Algorithm 2	N/A	O(n^2)	N/A	- n and T_i must be known - $T_1 \leq T_2 \leq \ldots \leq T_n$	- Algorithm 2 finds the number of harmonic chains
Algorithm 3	N/A	O(n^3)	N/A	- n and T_i must be known - $T_1 \leq T_2 \leq \ldots \leq T_n$	- Algorithm 3 finds a utilization bound

6 Characteristics of the Schedulability Conditions

In this section, Table 6 is introduced to show all schedulability conditions and their corresponding utilizacion bounds, time complexities, $lim_{n\to\infty}$, restrictions and tightness. It can be noted that algorithm 3 has the highest complexity among all conditions, while conditions L&L and UO have the lowest complexities. Recall that a tight test is the best possible test that can be found using that task set knowledge. Note that in some cases the authors of some tests did not provided a tightness proof.

7 Simulation Experiments

In this section, a performance analysis is conducted for the schedulability conditions previously discussed. In our experiments, the schedulability conditions were computed for each task set. The metric used in our experiments for obtaining the performance of the schedulability conditions is the *guarantee ratio*.

$$\text{Guarantee Ratio} = \frac{\text{number of task sets accepted}}{\text{total number of experiments}} \tag{22}$$

The guarantee ratio (GR) of schedulability condition A will be denoted as GR(A). A task set is *accepted* only if its schedulability condition yield positive results (i.e., all tasks meet the condition). The guarantee ratio is obtained by the ratio of the *experiments* (number of task sets) accepted and the *total number of experiments* (i.e., 1000). We are including for comparison the results of the experiments for the Le exact test.

7.1 Experiment 1

In the first set of experiments, shown in Figures 12, 13, 14, and 15, simulations were conducted as follows. Each data shown in the Figures, denote the average of the results obtained for 1000 simulations. On each simulation the task generation is as follows.

The periods of each task were generated following a uniform distribution with values in the range of 100 and 500. The execution times of the tasks were generated following a uniform distribution with values, $1 \leq C_i \leq \alpha\, T_i$, with $\alpha = 0.2$, $\alpha = 0.5$, $\alpha = 0.8$, and $\alpha = 1.0$. α denotes the maximum utilization U_i^{max} allowed for each task in the system, so for example when $\alpha = 0.5$, the utilization of each task u_i varies in the range $[0.01, 0.5]$. The number of tasks in each simulation varies depending on the total utilization U, which varies between 70% and 95%. Tasks are generated until $U^T = \sum_{i=1}^{n} u_i = U$ is achieved. When generating a new task τ_{n+1} with utilization u_{n+1}, if $U - 0.01 \leq (U^T + u_{n+1}) \leq U + 0.01$ tasks generation finishes. If $(U^T + u_{n+1}) \leq (U - 0.01)$ more tasks need to be generated, but when $(U^T + u_{n+1}) \geq (U + 0.01)$ task τ_{n+1} is discarded.

From Figure 12 we can observe that under $\alpha = 0.2$, conditions L&L, IP, UO, HC and ROOT, obtain similar results (Guarantee Ratio $\leq 2\%$). A comparison of Guarantee Ratios of the remaining conditions for $\alpha = 0.2$ is as follows: $GR(Le) > GR(A3) > GR(\text{DCT}) > GR(\text{A1}) > GR(SR) > GR(PO) > GR(RBOUND) > GR(ROOT)$.

From Figure 13 we can observe that under $\alpha = 0.5$, a comparison of Guarantee Ratios of the conditions is as follows: $GR(Le) > GR(A3) > GR(DCT) > GR(SR) > GR(A1) > GR(PO) > GR(RBOUND) > GR(UO) > GR(ROOT) > GR(IP) > GR(HC) > GR(L\&L)$.

From Figure 14 we can observe that under $\alpha = 0.8$, a comparison of Guarantee Ratios of the conditions is as follows: $GR(Le) > GR(DCT) > GR(SR) > GR(A3) > GR(A1) > GR(PO) > GR(RBOUND) > GR(UO) > GR(IP) > GR(ROOT) > GR(HC) > GR(L\&L)$.

From Figure 15 we can observe that under $\alpha = 1.0$, a comparison of Guarantee Ratios of the conditions is as follows: $GR(Le) > GR(DCT) > GR(SR) > GR(A3) > GR(A1) > GR(UO) > GR(IP) > GR(RBOUND) > GR(PO) > GR(ROOT) > GR(HC) > GR(L\&L)$.

The results obtained from comparing the inexact conditions in the first experiment are the following.

- From this experiment, it can be observed that L&L is the condition with worst performance. This is because other conditions use more information from the task set, allowing them to obtain better bounds. Condition L&L obtains better performance when utilization factor α increases.

This is because at higher utilizations, the number of tasks decreases, and L&L yield better bounds. The same behavior is observed in all schedulability conditions.

- Algorithm 3 is the inexact condition with best performance for $\alpha < 0.5$, but its performance is the third best for $\alpha \geq 0.5$.
- Algorithm DCT is the algorithm with second best performance for $\alpha < 0.5$ and the best for $\alpha \geq 0.5$.
- For all values of α algorithm 1 achieves a performance only worst than algorithms SR, DCT and algorithm 3.
- Algorithm SR achieves the second best performance for $\alpha \geq 0.5$, and fourth best performance for $\alpha < 0.5$. Note that, algorithms DCT and SR yield slightly better guarantee ratio than algorithms 1 and 3 for low values of utilization.
- The excelent performance of DCT and SR is explained from the fact that both algorithms transform the period set to another period set where all tasks belong to a single harmonic chain. However, it is clear that DCT always yield better performance than SR.
- Algorithm PO achieves the fifth best performance for $\alpha < 1.0$, and the eight best for $\alpha = 1.0$.
- Algorithm RBOUND achieves the sixth best performance for $\alpha < 1.0$, and the seventh best for $\alpha = 1.0$.
- Since PO and RBOUND compute and transform the ratio of the periods of the tasks their performance is better than other non-harmonic algorithms.
- Algorithm UO achieves the worst performance along with algorithms IP, ROOT, HC, and L&L for $\alpha = 0.2$, but seventh best performance for $0.5 \leq \alpha \leq 0.8$, and fifth best for $\alpha = 1.0$.
- HC and ROOT yield poor performance because few harmonic tasks are generated in the experiment. This can be explained as follows: In the case of HC, the size of the harmonic base is near the number of tasks. When α increases the number of tasks decreases, producing an increases in the performance of HC. The same situation occurs for ROOT, where the number of roots is also near the number of tasks. However note that ROOT always outperform algorithm HC.
- Algorithms ROOT, IP, HC and L&L are the algorithms with worst performance. In general the performance of ROOT is better than IP for values of $\alpha \leq 0.5$, and worst for $\alpha > 0.5$.
- Algorithm IP is always better than L&L because in its bound it uses more information from the task set (i.e., U_n and U_{n-1}). However its performance is not better than other algorithms.

7.2 Experiment 2

In the second set of experiments, shown in Figures 16, 17, and 18, there is a fixed number of tasks, 5, 10 and 20 tasks. C_i and T_i are computed as in the previous set of experiments, and α is set to 1.0, so the value of u_i varies in the range $[0.01, 1.0]$.

From Figure 16 (5 tasks) it is possible to note that in general the performance of the conditions is as follows: $GR(Le) > GR(DCT) > GR(SR) > GR(A3) > GR(A1) > GR(PO) > GR(RBOUND) > GR(UO) > GR(IP) > GR(ROOT) > GR(HC) > GR(L\&L)$. Note however that, in this case, UO yield better performance than RBOUND and PO for small values of utilization (i.e., $U \leq 75\%$). From

Figures 17 and 18 it is possible to note that the performance order of the conditions is similar as that observed in Figure 16 but with decreased performance.

From this experiment we conclude the following.

- SR and DCT are the algorithms with best performance. The reason for this excellent performance, as explained before, is because they transform the task sets to another set containing a single harmonic chain. This explains why they obtain good performance even when few harmonic chains are generated in this experiments. Furthermore, note that this effect is more notorious when the utilization of the task set is low, which means that they have more difficulty on transforming task sets with high utilizations.
- Algorithms L&L, IP, UO, RBOUND, PO, HC and ROOT yield poor performance (near zero) when the number of tasks is greater than 5.
- Condition HC only improves condition L&L. The poor performance of algorithm HC is because it considers the size of the harmonic base in its test, which in this experiments in most of the cases the size is almost equal than the number of tasks in the sets.
- Algorithms PO and RBOUND yield good performance under low number of tasks (i.e., less than 10 tasks). So, even when they compute period transformations, their performance is clearly affected by the number of tasks.
- ROOT is a condition based on the harmonicity of the periods, and it yields better performance than HC (and other algorithms) because it computes the number of *roots* in the task set, which in most of the cases is smaller than the size of the harmonic base (and consequently smaller than the number of tasks). In any case, the poor performance of HC and ROOT is explained because very few harmonic chains are generated in this experiment.

7.3 Experiment 3

The third set of experiments was conducted for testing the performance of the conditions considering variations in the period ratio. The generation of tasks did not considered the generation of harmonic chains in the task sets.

In this experiment all schedulability tests and algorithms followed the algorithm for task generation algorithms described in [Bini04, 2004]. Bini $et - al$ [Bini04, 2004] argued that traditional evaluation metrics and methods used for random generation of tasks parameters are greatly biased affecting the overall performance of the experiments. For these reasons they provided some efficient algorithms for generating task parameters.

Algorithm *UniFast* [Bini04, 2004] requires as input the number of tasks and the total utilization U, and provides the utilization of each task in the system, u_i.

This experiment (shown in Figures 19 to 22) was conducted for 5 tasks following the *UniFast* algorithm [Bini04, 2004].

Each data on the Figures, denotes the average of a set of 1000 experiments. The total utilization, U, is varied from 70% to 95% in steps of 5%. For each utilization value U, 1000 task sets where generated. The values of u_i for a given task on any task set was computed using the *UniFast* algorithm considering a given value of r. Once this utilizations are computed for each experiment, the period T_i of each task is computed following a uniform distribution with values in the range $[50, 1000]$, and $C_i = T_i/u_i$.

We generated 4 cases, each one with the same 1000 tasks sets, but with different values of r. In this test the ratio of the periods, r, is varied in values of $r \leq 2$, $r \leq 4$, $r \leq 8$, and $r \leq 16$. r is computed as $r = maxT_i/minT_i$ (for $i = 1, ..., n$). For any task set, the periods of the tasks were computed following

a uniform distribution in the range $[10, 20]$, $[10, 40]$, $[10, 80]$, and $[10, 160]$ for values of r equal to $2, 4, 8, 16$ respectively.

From this test we conclude the following.

- From Figures 19 to 22 it can be noted that the performance of the algorithms for 5 tasks is as follows: $GR(Le) > GR(DCT) \geq GR(SR) > GR(PO) > GR(RBOOUND) > GR(UO) > GR(ROOT) > GR(IP) > GR(HC) > GR(L\&L)$. However, for low utilization values (i.e., 75 %) the performance of ROOT, HC and UO, is greater than the performance of algorithms IP, PO and RBOUND.

- In general, it can be observed that algorithm L&L, IP, UO do not change their performance with higher period ratios. The reason for this behavior is that their conditions do not depend on the tasks's periods.

- Algorithms PO, RBOUND, SR and DCT increase slightly their performance with higher period ratios. Algorithms PO compute the difference between two periods in the set. So, it computes how close from being harmonic, are any 2 periods in the set. When r is higher PO finds an smaller value of β (see condition PO), so it finds that the tasks are closer from being harmonic, and consecuently it yields better performance.

  Algorithm RBOUND tranforms the task set into another task set where the period ratio is in the range [1,2]. So, when the value of r increases the task set resulting from scaletaskset tend to move towards 1, yielding better performance.

  The excelent performance of DCT and SR is explained because they transform the task sets to another set containing a single harmonic chain.

- Algorithms HC and ROOT show an slight decrease on their performance with higher period ratios. So, for these algorithms it is clear than lower harmonic chains are generated with higher period ratios, expalining the decrease in their performance.

- We executed similar experiments for 10 and 20 tasks (not showed in the paper), but their behavior was similar to that observed for 5 tasks, although with decreased performance.

7.4 Experiment 4

The fourtht set of experiments was conducted for testing the performance of the conditions considering the generation of harmonic chains. As discussed before, in previous experiments the generation of tasks did not considered the generation of harmonic chains in the task sets. Each data on the Figures, denotes the average of a set of 1000 experiments.

In this experiment tasks were generated using the UniFast Algorithm, as in Experiment 3, and two tests were conducted.

First Test. The first test (shown in Figures 23 to 34) was conducted considering a given percentage (20%, 40%, 60% and 80%) of harmonic task for 5, 10 and 20 tasks. That is for each experiment, a given percentage of tasks is harmonic and tasks are formed from a single harmonic chain. While generating such single harmonic chain, the periods of the tasks T_i (for $i = 2, ..., n$) are transformed as follows. $T_i = x \ T_{i-1}$ where x is a number obtained from a uniform distribution in the range $[2, 9]$.

From Figure 23 (5 tasks) note that DCT and SR are the best algorithms, followed by UO, PO, IP, RBOUND, ROOT, HC and L&L. However, note that ROOT and HC yield better performance than IP, PO, and RBOUND only for small utilization values (i.e., 75 %).

From Figure 24 (5 tasks) note that DCT and SR and the best algorithms, followed by PO, RBOUND, ROOT, UO, HC, IP, and L&L. However, note that ROOT, HC and UO, yield better performance than PO, RBOUND and IP only for small utilization values (i.e., 75 %).

From Figure 25 (5 tasks) we note again that algorithms DCT and SR yield the best performance, followed by RBOUND, ROOT, UO, HC, IP and L&L. However, note that algorithms ROOT, HC, and UO yield near 100 % guarrantee ratio for low utilization values.

From Figure 26 note that the performance of algorithms is similar to the one observed in Figure 25.

Figures 27 to 30 indicate again that algorithms DCT and SR are the algorithms with best performance. From this figures it can be observed that in general the performance of the algorithms is as follows: $GR(Le) > GR(DCT) > GR(SR) > GR(PO) \geq GR(RBOUND) \geq GR(ROOT) \geq GR(HC) \geq GR(IP) \geq GR(L\&L)$. However, UO tend to yield better performance than PO and RBOUND for low utilization values and when the percentage of harmonic tasks is low (20 % and 40 %).

From Figures 31 to 34 note that the performance of most of the algorithms is zero. Only algorithms DCT and SR yield good performance under any percentage of harmonic tasks. In this case, only algorithms PO, RBOUND and ROOT yield performance higher than zero when 80 % of the tasks are harmonic.

From this test we conclude the following:

- DCT and SR always yield the best performance.
- Algorithms ROOT, HC and UO yield high performance for low utilization values (less than 80 %), but their performance decreases sharply at high utilization values. Their performance is quite poor for number of tasks greater than 5.
- In general, we observe that for most of the cases, $GR(PO) > GR(RBOUND) > GR(IP) \geq GR(L\&L)$.
- L&L, IP and UO does not change their performance with higher percentage of harmonic tasks. On the other hand, there is a clear increase in the performance of DCT, SR, PO, ROOT, HC and RBOUND when higher harmonic tasks are included in the tasks sets.
- In general, it is notorious the decrease in performance when the number of tasks increases for all algorithms, except for algorithms DCT and SR.

Second Test. In the second test (shown in Figures 35 to 37) the number of tasks is fixed (10 tasks) and the number of harmonic chains is varied. In this test, any task belong to only one harmonic chain and the number of harmonic chains generated was 2, 4 and 6. The number of tasks on each harmonic chain is computed as $\lfloor$(number of tasks / number of chains)$\rfloor$. The remaining tasks are included into the last harmonic chain. In the generation of each harmonic chain the periods of the tasks is transformed as follows. An initial period is computed following a uniform distribution in the range $[50, 1000]$, the following periods in the chain are multiplied by a unique integer uniformly distributed in the range $[2, 9]$.

From this test we conclude the following.

- Results from Figures 35 to 37 indicate that DCT and SR are the algorithms with best performance.
- ROOT and HC yield 100 % performance for 2 harmonic chains and utilization less than 85 %, and for 4 harmonic chains and utilization less than 80 %. However, their performance decreases sharply after those values of utilization.

- From Figure 35 we note that, for utilizations higher than 80 % the performance of the algorithms is as follows: $GR(Le) > GR(DCT) > GR(SR) > GR(PO) > GR(RBOUND) > GR(ROOT) > GR(UO) > GR(IP) > GR(HC) > GR(L\&L)$.
- From figure 36 we note that, for utilizations higher than 75 % the performance of the algorithms is as follows: $GR(Le) > GR(DCT) > GR(SR) > GR(PO) > GR(RBOUND) > GR(ROOT) > GR(UO) > GR(IP) > GR(HC) > GR(L\&L)$.
- From Figure 37 we note that the performance of the algorithms is as follows: $GR(Le) > GR(DCT) > GR(SR) > GR(PO) > GR(ROOT) > GR(RBOUND) > GR(UO) > GR(IP) > GR(HC) \geq GR(L\&L)$.
- It is important to note that in this test algorithms L&L, IP and UO yield the same performance on all experiments.

8 Conclusions

Many real-time applications demand efficient and low cost schedulability tests for on-line admission control. In this paper we survey the best known exact and inexact schedulability tests 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 are evaluated for different utilization factors, α values and number of tasks. Additional experiments were conducted considering task sets generated with harmonic periods. In this experiments, most of the algorithm showed an increased performance.

We believe that the decision of choosing one schedulability tests or another for a particular real-time application should depend not only on its performance, but also it must consider the allowed computational complexity.

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

References

[Andersson, 2001] B. Andersson, Sanjoy Baruah and Jan Jonsson, "Static-priority Scheduling on Multiprocessor", *Proceedings of the IEEE Real-Time Systems Symposium*, London, England, December 2001.

[Abdelzaher, 2004] Abdelzaher, T.F., Sharma, V., and Lu, C., "A Utilization Bound for Aperiodic Tasks and Priority Driven Scheduling," *IEEE Transactions on Computers*, 53(3): 334-350, March 2004.

[Audsley, 1993] Audsley, N. C., Burns, A., Tindell, K., and Wellings, A. "Applying New Scheduling Theory to Static Priority Preemptive Scheduling", *Software Engineering Journal*, vol. 8, no. 5, pp. 284-292, 1993.

[Bini, 2001] Bini, E., Buttazzo. G.C., and Buttazzo, G. "A Hyperbolic Bound for the Rate Monotonic Algorithm", *In IEEE Proc. of the 13th Euromicro Conf. on Real-Time Systems*, pp. 59-66, 2001.

[Bini04, 2004] Bini, E. and Buttazzo. G.C. "Biasing Effects in Schedulability Measures", *In IEEE Proc. of the 13th Euromicro Conf. on Real-Time Systems*, 2004.

[Burchard, 1995] Burchard, A., Liebeherr, J., Oh. Y., and Son, S.H. "New Strategies for Assigning Real-Time Tasks to Multiprocessor Systems", *IEEE Transactions on Computers*, vol. 44, number 12, pp. 1429-1442, December 1995.

[Burns, 1996] A. Burns, R. Davis and S. Punnekkat, "Feasibility Analysis of Fault-Tolerant Real-Time Task Sets", *In IEEE Proceedings of the Euromicro Workshop on Real-Time Systems*, 29-33, 1996.

[Lipari, 2000] G. Lipari, G. Buttazzo, "Schedulability Analysis of Periodic and Aperiodic Tasks with Resource Constraints", *J. of Systems Architecture, (46) 2000.*

[Buttazzo, 1998] G.C. Buttazzo, "Red: A Robust Earliest Deadline Scheduling Algorithm", *Proc. of Third Int. Workshop on Responsive Computing Systems*, Spain, Dec. 1998.

[Chen, 2003] Chen, D., Mok, A.K., and Kuo, T. "Utilization Bound Revisited", *IEEE Transactions on Computer*, vol. 52, No. 3, pp. 351-361, March 2003.

[Dhall, 1978] Dhall, S.K., and Liu, C.L. "On a Real-Time Scheduling Problem", *Operations Research*, vol. 26, number 1, pp. 127-140, 1978.

[Han, 1997] Han, C.C., and Tyan, H.Y. "A Better Polynomial-Time Schedulability Test for Real-Time Fixed-Priority Scheduling Algorithms", *Proc IEEE 18th Real-Time Systems Symp.*, pp. 36-45, 1997.

[Joseph, 1986] Joseph, M., and Pandya, P. "Finding Response Times in a Real Time System", *The Computer Journal. British Computer Society*, vol. 29, no. 5, pp. 390-395, 1986.

[Kuo, 1991] Kuo, T., and Mok, A.K. "Load Adjustment in Adaptive Real-Time Systems", *Proc. IEEE Real-Time Systems Symp.*, pp. 160-171, 1991.

[Kuo, 2000] Kuo. T., Liu, Y., and Lin, K. "Efficient On-Line Schedulability Tests for Priority Driven Real-Time Systems", *Sixth IEEE Real Time Technology and Applications Symposium (RTAS 2000)*, May 31 - June 02, pp. 4-13, 2000.

[Lauzac, 2003] Lauzac, S., Melhem, R., and Mosse, D. "An Improved Rate-Monotonic Admission Control and its Application", *IEEE Transactions on Computers.* vol. 52. No. 3, pp. 337-350, March 2003.

[Lauzac: 2000] S. Lauzac, "Multiprocessor Scheduling of Preemptive Periodic Real-Time Tasks with Error Recovery", *PhD. Thesis. Computer Science Department, University of Pittsburgh*, 2000.

[Lee, 2004] Lee, C.-G., Sha, L., and Peddi, A., "Enhanced Utilization Bounds for QoS Management", IEEE Transactions on Computers, Vol. 53, No. 2, Feb. 2004

[Lehoczky, 1989] Lehoczky, J.P., Sha, L., and Ding, Y. "The Rate-Monotonic Scheduling Algorithm: Exact Characterization and Average Behavior", *Proceedings of the IEEE Real-Time Systems Symposium*, pp. 166-171, 1989.

[Lehoczky, 1990] Lehoczky, J.P. "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines", *in Proc. IEEE, Real-Time Systems Symposium*, pp, 201-209, 1990.

[Leung, 1982] Leung, J. Y.-T., and Whitehead, J. "On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks, *Performance Evaluation*, number 2, pp. 237-250, 1982.

[Liu, 1973] Liu, C.L., and Layland, W. "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment", *Journal of the ACM*, vol. 20, number 1, pp. 46-61, January 1973.

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

[Liu, 2002] Liu H., and Hu X., "Processor utilization bounds for real-time systems with precedence constraints", *Design Automation for Embedded Systems, An International Journal*, v7, p89-114, September, 2002.

[Lopez, 2003] López, J.M., García. M., Díaz J.L., and García D.F., "Utilization Bounds for Multiprocessor Rate-Monotonic Scheduling", *Real Time Systems*, vol. 24, Issue 1, January 2003.

[Oh, 1995] Oh, Y., and Son, S.H. "Fixed Priority Scheduling of Periodic Tasks on Multiprocessor Systems", *Tech. Report CS-95-16, Univ. Of Virginia. Dept. of Computer Science*, March 1995.

[Park, 1996] Park, D.W., Natarajan, S., and Kanevsky, A. "Fixed Priority Scheduling of Real-Time Systems using Utilization Bounds", *Journal of Systems and Software*, Elsevier. Vol. 33, pp. 57-63, 1996.

[Rajkumar, 1991] R. Rajkumar, "Synchronization in Real Time Systems: A priority inheritance Approach", *Kluwer Academic Publishers*, 1991.

[Sha, 1990] L. Sha, R. Rajkumar and J. P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization", *IEEE Transactions on Computers*, 39(9):1175-1185, 1990.

[Sprunt, 1989] B. Sprunt, L. Sha and J. Lehoczky, " Aperiodic Task Scheduling for Hard Real Time Systems", *Journal of Real-Time Systems*, pp. 27-60, 1989.

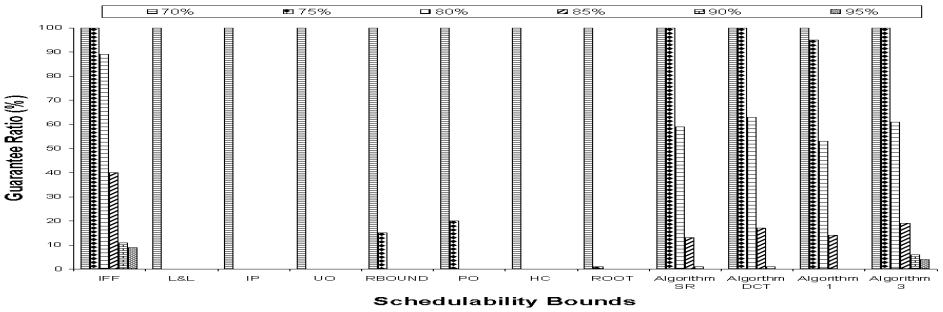

Figure 12: Guarantee ratio with $\alpha = 0.2$

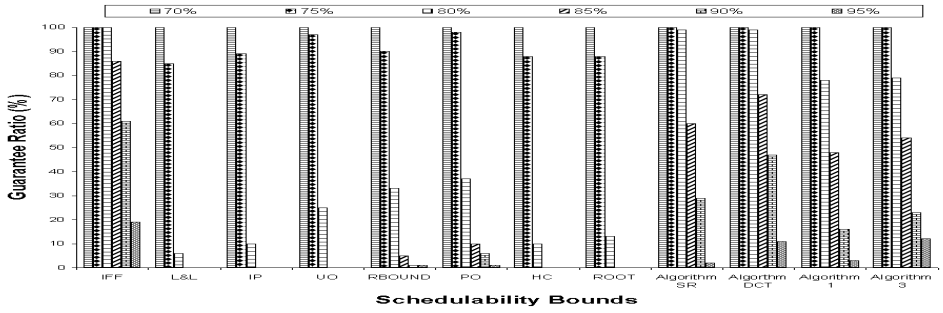

Figure 13: Guarantee ratio with $\alpha = 0.5$

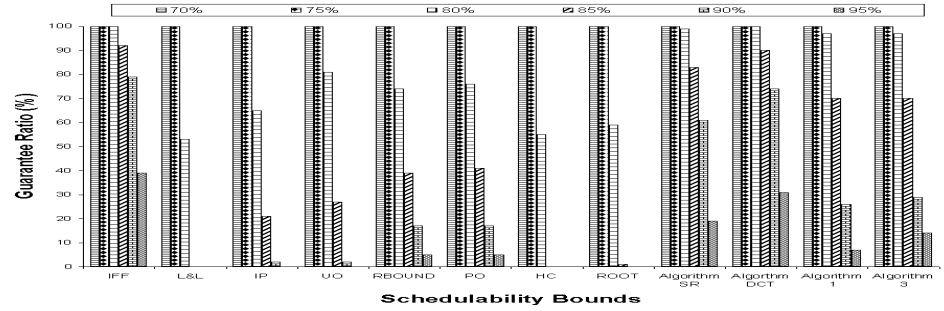

Figure 14: Guarantee ratio with $\alpha = 0.8$

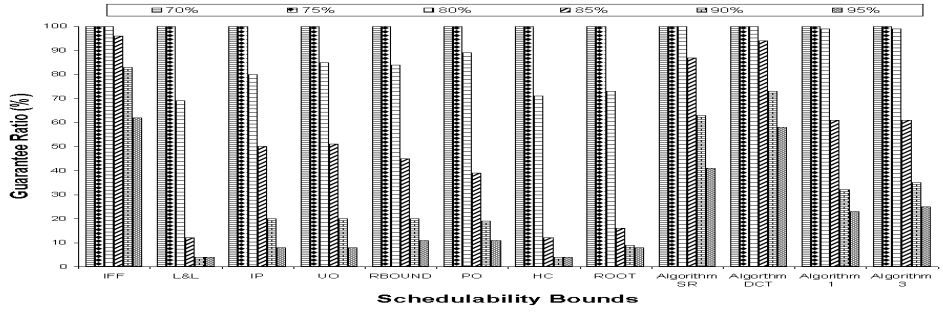

Figure 15: Guarantee ratio with $\alpha = 1.0$

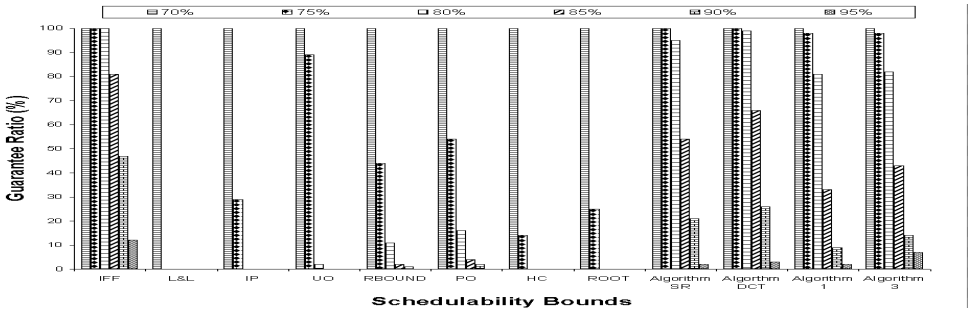

Figure 16: Guarantee ratio with 5 Tasks and $\alpha = 1.0$

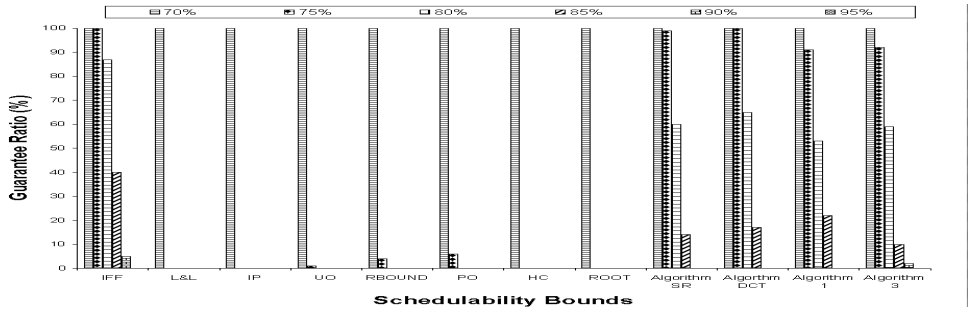

Figure 17: Guarantee ratio with 10 Tasks and $\alpha = 1.0$

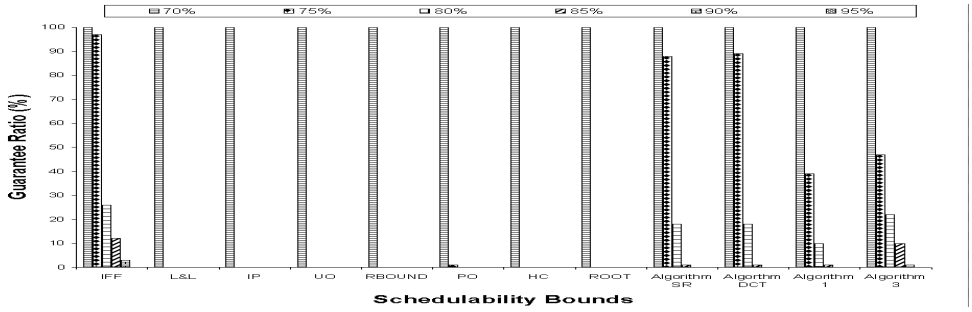

Figure 18: Guarantee ratio with 20 Tasks and $\alpha = 1.0$

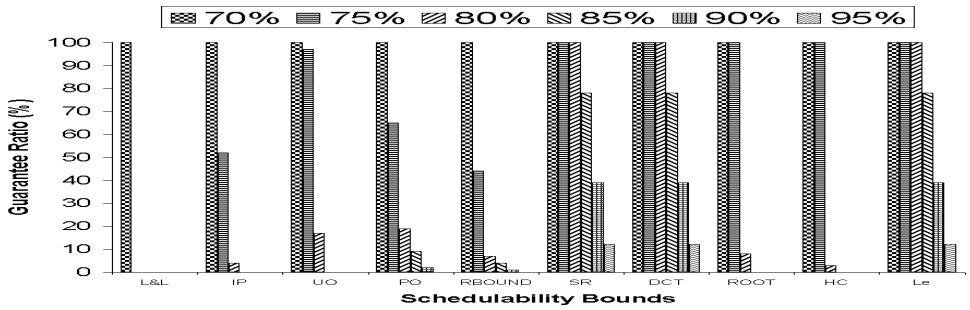

Figure 19: Test with 5 tasks and ratio of periods $r \leq 2$

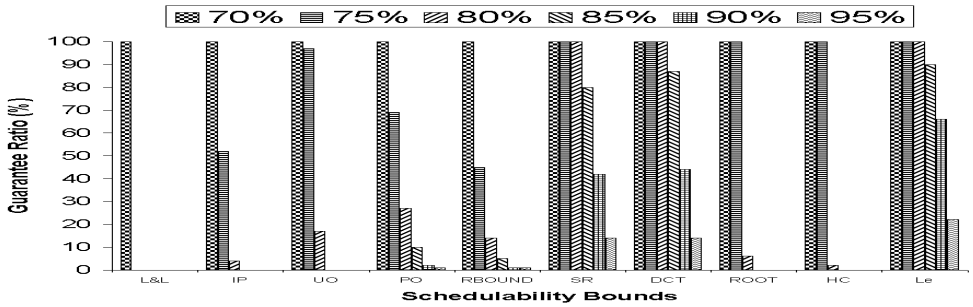

Figure 20: Test with 5 tasks and ratio of periods $r \leq 4$

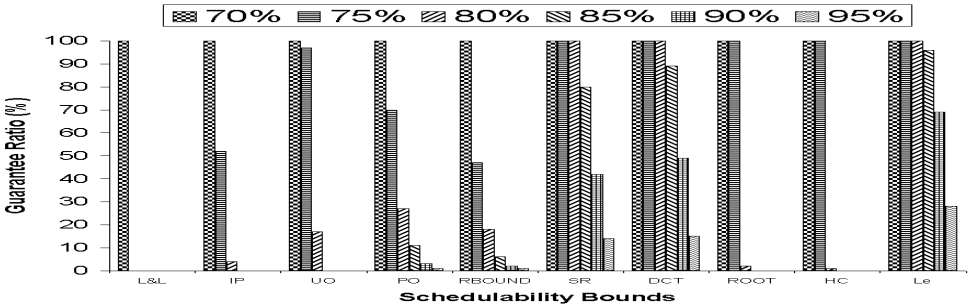

Figure 21: Test with 5 tasks and ratio of periods $r \leq 8$

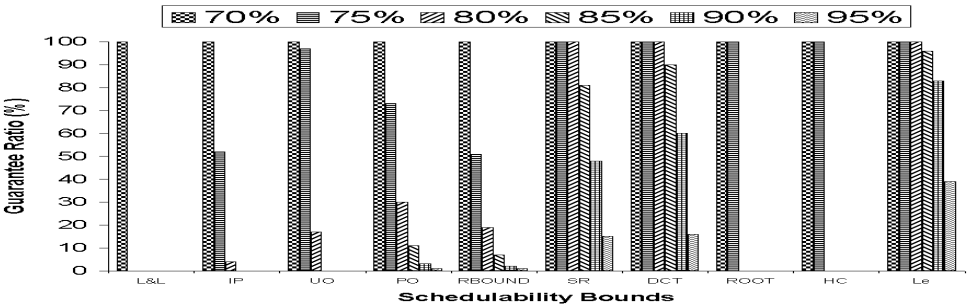

Figure 22: Test with 5 tasks and ratio of periods $r \leq 16$

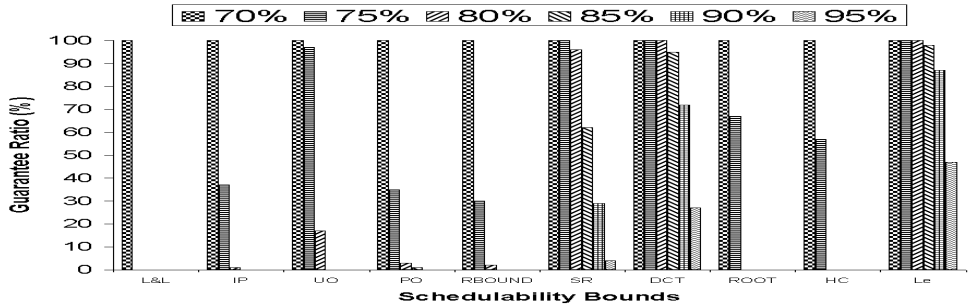

Figure 23: Harmonic Chains Test: with 5 tasks and 20 % of harmonic tasks

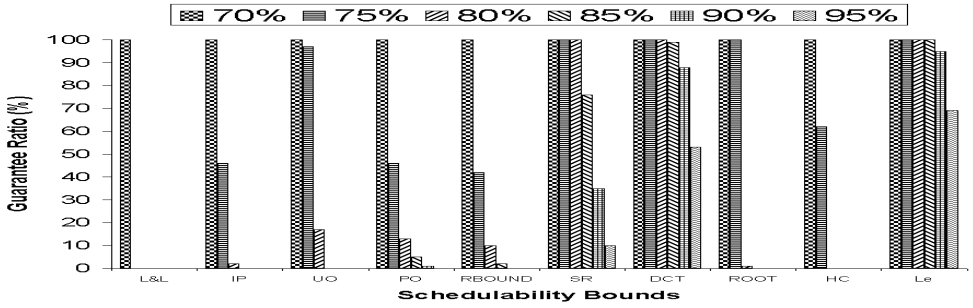

Figure 24: Harmonic Chains Test: with 5 tasks and 40 % of harmonic tasks.

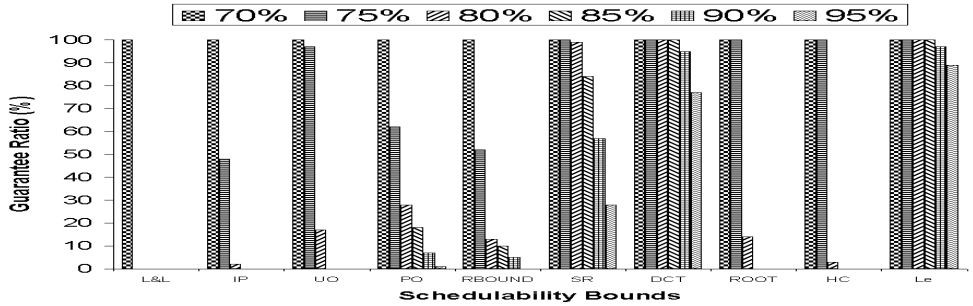

Figure 25: Harmonic Chains Test: with 5 tasks and 60 % of harmonic tasks.

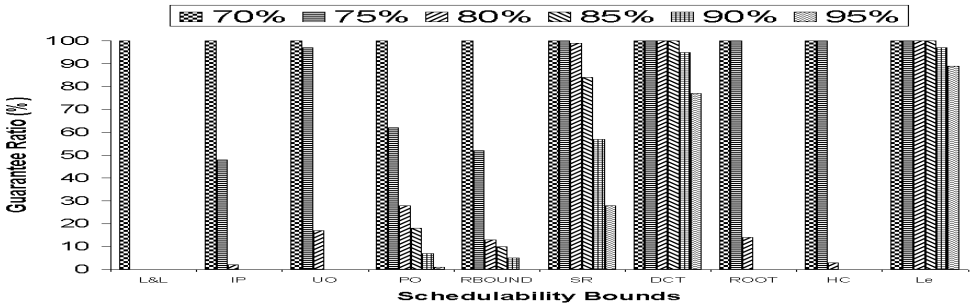

Figure 26: Harmonic Chains Test: with 5 tasks and 80 % of harmonic tasks.

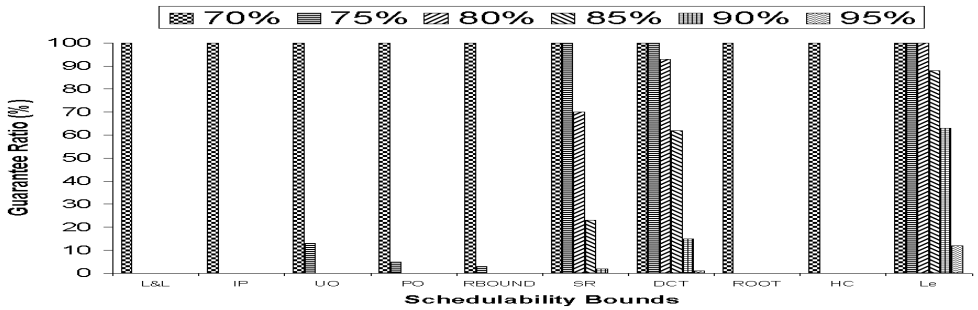

Figure 27: Harmonic Chains Test: with 10 tasks and 20 % of harmonic tasks.

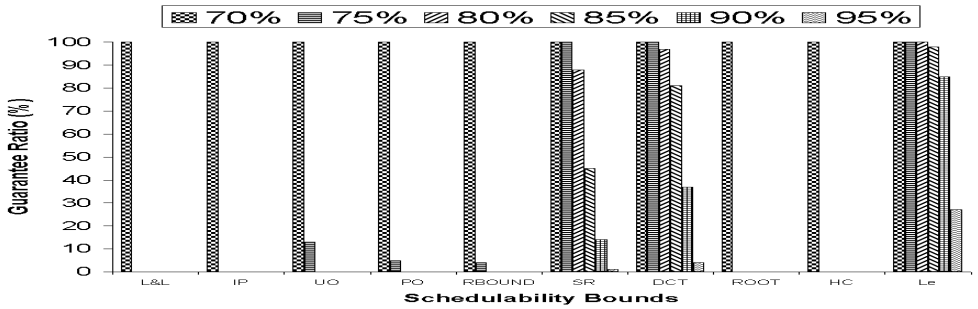

Figure 28: Harmonic Chains Test: with 10 tasks and 40 % of harmonic tasks.

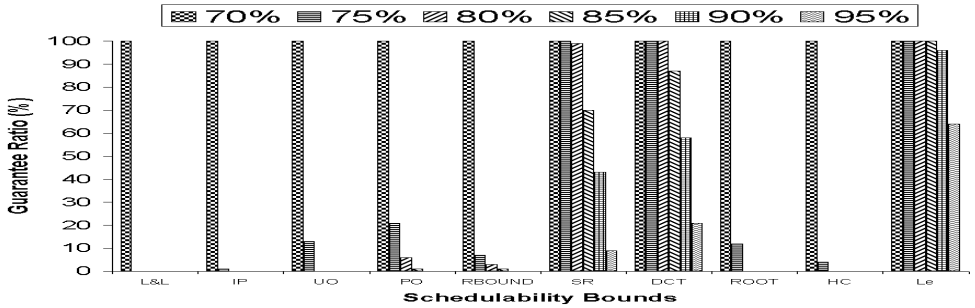

Figure 29: Harmonic Chains Test: with 10 tasks and 60 % of harmonic tasks.

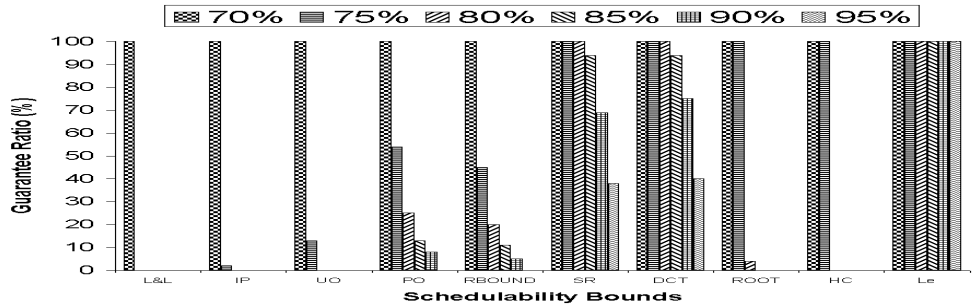

Figure 30: Harmonic Chains Test: with 10 tasks and 80 % of harmonic tasks

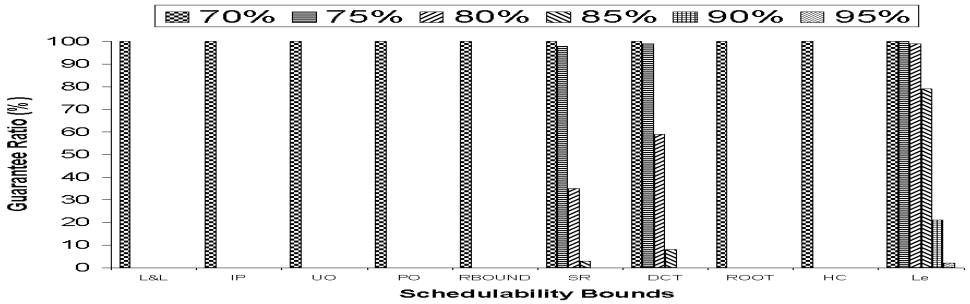

Figure 31: Harmonic Chains Test: with 20 tasks and 20 % of harmonic tasks

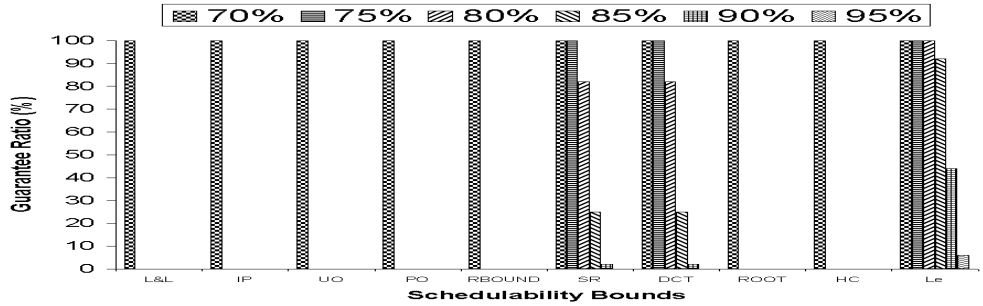

Figure 32: Harmonic Chains Test: with 20 tasks and 40 % of harmonic tasks.

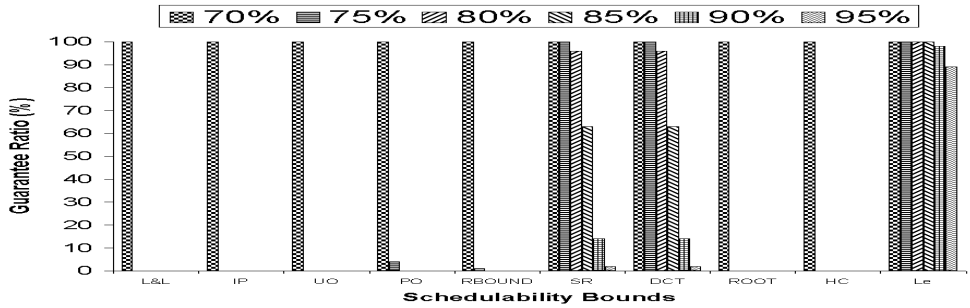

Figure 33: Harmonic Chains Test: with 20 tasks and 60 % of harmonic tasks.

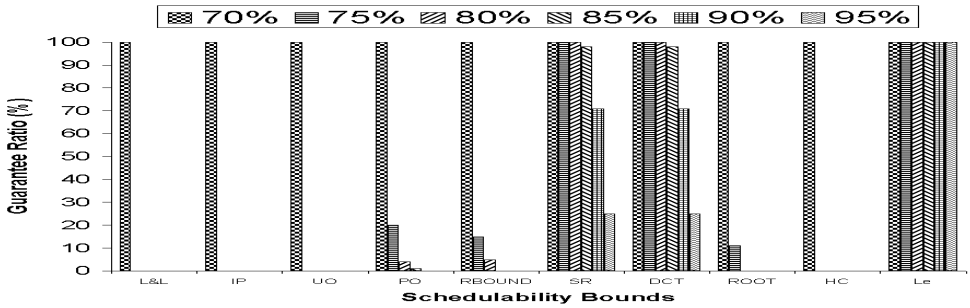

Figure 34: Harmonic Chains Test: with 20 tasks and 80 % of harmonic tasks.

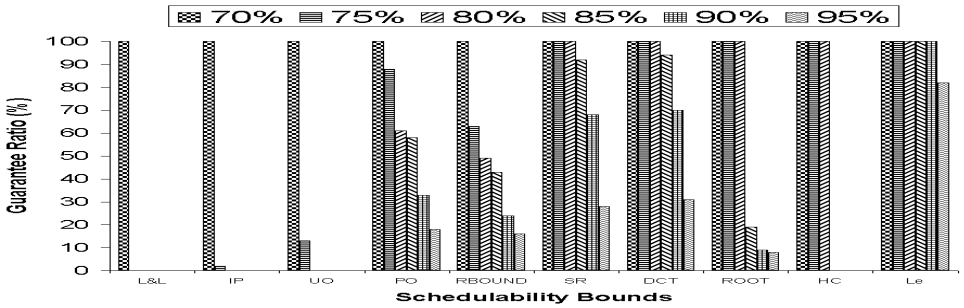

Figure 35: Harmonic Chains Test: with 2 harmonic chains

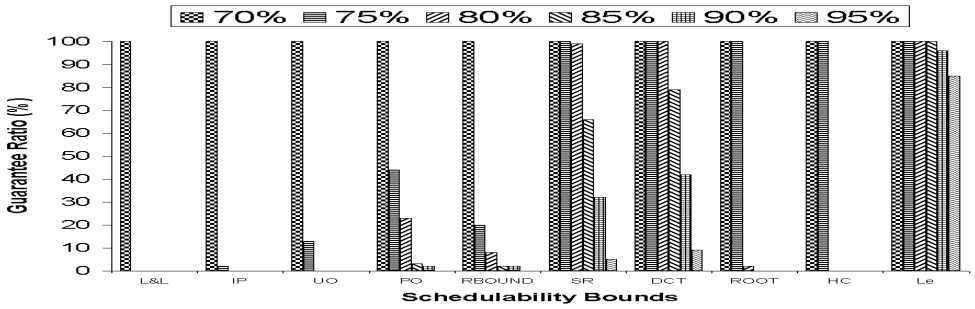

Figure 36: Harmonic Chains Test: with 4 harmonic chains

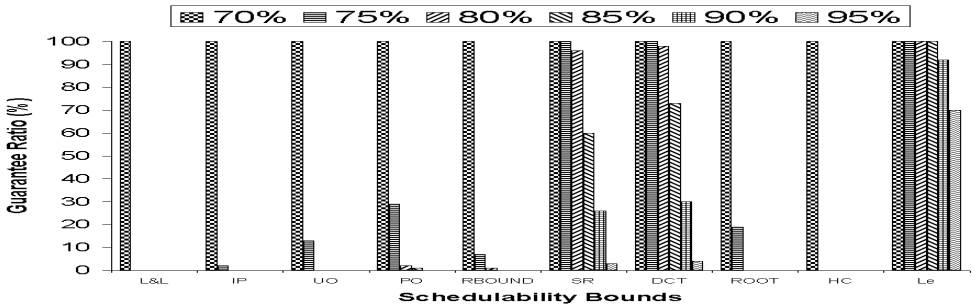

Figure 37: Harmonic Chains Test: with 6 harmonic chains