A Contribution to the Deadline Monotonic Scheduling Theory*

Sibelius Lellis Vieira
Instituto de Informática
Universidade Federal de Goiás
Goiânia. GO. Brasil 74001-970
e-mail:sibelius@inf.ufg.br

Maurício Ferreira Magalhães
Dept. de Engenharia de Computação e Automação Industrial
Universidade Estadual de Campinas
Campinas. SP. Brasil 13081-970

Abstract

In this paper a new schedulability test is proposed for a periodic task set when task deadlines are arbitrary, provided they are not larger than their respective periods. This task set model can model a system of jitter constrained process found in multimedia systems. The test is based on a number of evaluation tests using the periodic task set utilization factor. It is shown that when the task deadline is not much smaller than its period, the results provided can give the best results among the ones found in the literature, regarding efficiency and computational costs.

keyword list: Operating Systems, Optimization, Engineering Computation.

1 Introduction

Today's Hard Real-Time Systems(HRTS) must be designed with the goals of flexibility and adaptiveness required in a dynamic environment, while keeping a deterministic behaviour[1]. By a dynamic environment we mean a environment composed of dynamic tasks, that can arrive and disappear constantly. Flexibility and Predictability (deterministic behaviour) can be achieved through a carefully scheduling policy. Concerning the scheduling problem such HRTS must assure that the schedulability tests are fast, so that they may be applied on-line. These schedulability tests can only be applied if they are (1) predictably fast (2) provide a convenient utilization factor[2].

According to the Deadline Monotonic Theory Scheduling[3], the optimal priority association is known as the Deadline Monotonic priority association, provided that tasks are synchronous and deadlines have arbitrary values. The Deadline Monotonic Scheduling Theory has greatly evolved during the past years, due to its considerable amount of applicability and the benefits it is providing to the field of Hard Real-Time Systems[4]. This scheduling policy asserts higher priorities to tasks with smallers deadlines. We want to provide a utilization factor bound that allows us to assure that the periodic task set is schedulable[5]. A lot of schedulatility tests have been proposed in these years, among them we cite the simulation of arbitrary deadlines as blocking[6], the uniform deadline variation[7] and the interference analysis[8]. We propose a test named arbitrary deadline variation approach(ADVA), which can lead to a suitable tradeoff between performance and computational costs.

In short, we address the issue of real-time CPU scheduling in hard real-time operating systems. The approach we propose is to design a new admission test for periodic tasks which guarantee that a run time

*This work has been partially funded by FUNAPE/UFG

scheduler, based on the Deadline Montonic priority, will be able to honour all the specified deadlines, where the deadlines may be earlier than the end of the current period. These tasks are sometimes refered as jitter constrained tasks, as they show up less jitter than conventional periodic tasks[9]. This set model is very suitable for multimedia applications where jitter issues are a major concern[10].

In section 2, the related work is reviewed. In section 3, we illustrate our schedulability test, using a two-task system. In section 4 we present the results for a n-task system. We present comparisons among the methodologies in section 5 and final remarks in section 6.

2 Related Work

We present a review with the main results concerning the schedulability of a periodic task set when arbitrary deadlines are present, provided they are smaller than the correspondent periods[11]. The conditions evaluated to guarantee that a task set is schedulable are based on sufficient or exact tests. A sufficient test assures that a set is schedulable if some condition is true. However, if the condition is false the set might or might not be schedulable. Generally, the advantage of the sufficient condition is that it may be evaluated in polynomial time[3]. On the other hand, it may also give poor results compared with the exact condition. In HRTS speech fast and predictable responses are a valuable feature. We show in this section some approaches which lead to sufficient tests. They differ mainly in their scheduling efficiency, defined as the CPU relative utilization of schedulable tasks. Let us explain some terminology first: we refer to one periodic task as $T_i = (c_i, d_i, p_i)$, where c_i is the execution time, d_i is the deadline and p_i its period. All tasks are synchronous, implying they are activated at the same time. When $d_i = p_i$, it is proven that if $U \leq n(2^{1/n} - 1)$ the task set $T = \{T_1, ... T_n\}$ is schedulable[5], and U is known as the utilization factor of the set, $U = \sum_{i=1}^{n}(c_i/p_i)$.

2.1 Simulation Arbitrary Deadline as Blocking

Given a task set $T = \{T_1, ... T_n\}$, where $T_j = (c_j, d_j, p_j)$, it is assumed that if for task T_j we have $d_j < p_j$, then it has an associated blocking, named B_j[6]. Take $k_i = p_i/d_i$. The task set is schedulable if:

$$\sum_{i=1}^{n}(c_i/p_i) + max(B_i/p_i) \leq n(2^{1/n} - 1)$$

where $B_i = p_i - d_i$. Then, we get that if $U \leq n(2^{1/n} - 1) - max(1 - 1/k_i)$ the set is schedulable.

This result applies to the set T in its entirety. Eventually, this sort of results can be improved by noticing that T is schedulable if all its subsets also are. However, we might have subsets of T schedulable, even if T is not schedulable. Let $l < n$, and if the condition above is valid so is $\sum_{i=1}^{l}(c_i/p_i) + max(B_i/p_i) \leq l(2^{1/l} - 1)$.

2.2 Uniform Deadline Variation Approach

The uniform deadline variation approach(UDVA) constraints the arbitrary deadline values to be uniformly smaller than the respective periods, in other words, $p_i/d_i = p_j/d_j$ whatever i and j. Being $k = p_i/d_i$, the sufficient condition for successfully scheduling the set is [7,12]:

- If $k \leq 2$ and $U \leq n((2/k)^{1/n} - 1) + (1 - 1/k)$
- If $k \geq 2$ and $U \leq k$

Again, the sufficient condition might be applied to a subset of T, let us say with l tasks, leading to the condition $\sum_{i=1}^{l}(c_i/p_i) \leq l((2/k)^{1/l} - 1) + (1 - 1/k)$.

2.3 Interference Analysis

In this approach, it is possible to apply several sufficient conditions, differing among themselves in the way they evaluate the interference factor[8]. This method is based on that higher priority tasks will interfere with the execution of lower priority ones. The evaluated amount of this interference is variable, ranging from a very simple activation counting of one high priority task in the deadline of a low priority task, until more sofisticated evalutions of interference. Let I_j be the interference of tasks with higher priority than T_j, the schedulability of T_j is guaranteed if:

$$c_j + I_j \leq d_j$$

We shall proceed with the evaluation of the tests for all tasks. The computational cost depends of how I_j is evaluated. For instance, if $I_i = \sum_{j=1}^{i-1} \lceil d_i/p_j \rceil c_j$, the complexity is $O(n^2)$. As the approach we propose is $O(n)$, since is it based only at the utilization factor, we are not considering this approach for comparixon.

3 Feasible Schedule for a Two-task System using the Arbitrary Deadline Variation Approach

We now begin our investigation of finding a feasible schedule for a two-task system, using an approach we name arbitrary deadline variation approach(ADVA). Given a two-task system $T = \{T_1, T_2\}$, such that $T_i = (c_i, d_i, p_i)$, where $c_i \leq d_i \leq p_i$, being c_i the execution time, d_i the relative deadline and p_i the period of task T_i, we want to know if the task set is schedulable through a fixed priority association. We refer to this problem as the schedulability question. It is known that when the deadline of each task is not greater than its period, the deadline monotonic is the optimal priority association[3].

To find out a solution for the schedulabilily question that can be stated as a sufficient test, we need to provide an evaluation of the minimum utilization factor(U_{min}). Its definition is such that if the task set utilization factor is not greater than U_{min} then the set is schedulable. Let us assume $d_1 \leq d_2$ and $d_i = p_i$, for both tasks. We know that the utilization factor(U) is given by:

$$U = c_1/p_1 + c_2/p_2$$

The 0.83 limit is known as the minimum utilization factor that guarantees T schedulability, applying the rate monotonic priority association[5]. This means that any two-task set T with $U_T \leq 0.83$ is schedulable through the rate monotonic scheduling. It is possible to show that there is a system T, with $U_T = 0.84$, which is not schedulable. For the 0.83 limit, the task parameters are:

$$d_2 = p_2 = 1.414p_1 = 1.414d_1$$
$$c_1 = p_2 - p_1 \text{ e } c_2 = p_1 - c_1$$
$$U = 2(2^{1/2} - 1) = (2^{1/2} - 1) + (2^{1/2} - 1)$$

We notice that the minimum utilization factor occurs when we uniformly distribute the CPU time among the competing tasks, such that each task has the same amount of utilization.

We show in Figure 1 the CPU utilization for T given by $T_1 = (414, 1000)$ and $T_2 = (586, 1414)$. The schedulability is guaranteed once we have $U = 0.828$. When we increase the execution time of T_2 in one unity, say $T_2 = (587, 1414)$, the system is no longer schedulable. This missing deadline is shown in Figure 2.

For the general case we name $k_i = p_i/d_i$ the squeeze factor of task T_i. The greater the squeeze factor the harder will be to meet the task timing requirements. The minimum squeeze factor is 1, indicating that the task must finish at any time within its period. For the general case $k_i \geq 1$. Our initial goal is to analyze how the squeeze factor will affect the schedulability of the two-task system. We regard some specific cases:

- $k_1 \geq 1$ e $k_2 = 1$

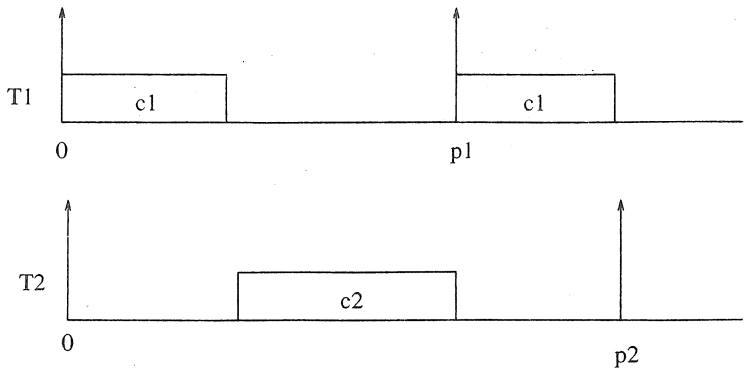

Figure 1: T_1, T_2, valid schedule

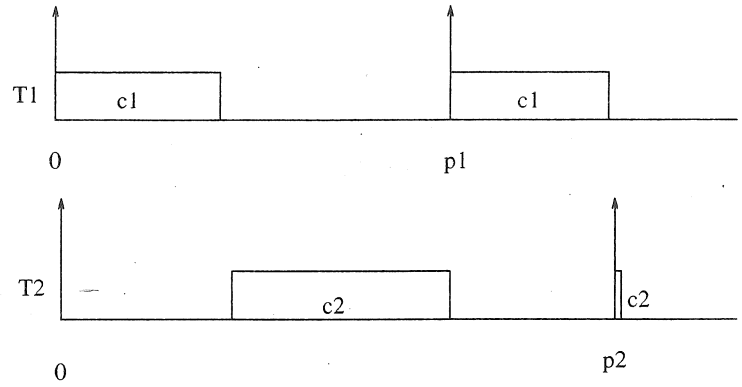

Figure 2: T_2 missing deadline

- $k_1 = 1$ e $k_2 \geq 1$
- $k_i \geq 1$

3.1 Specific scenarios

As our first specific scenario, let us analyze the behaviour ot the tasks in T when $k_2 = 1$ e k_1 is arbitrary. As $d_2 = p_2$, task T_2 would not miss its deadline if $U \leq 0.828$, so its schedulability is not affected. Regarding T_1, it is possible to miss a deadline if $d_1 \leq 0.828 p_1$ and $c_1 = 0.828 p_1$. So we have a new schedulability condition that relates to k_1. However if $d_1 > 0.828 p_1$ the previous condition is kept. Thus T_1 is schedulable if $U_1 \leq U_{min}(1)$, where $U_i = c_i/p_i$ and $U_{min}(1) = 1/k_1$. On the other hand T_2 will be schedulable if $U_1 + U_2 \leq 0.828$, and T is schedulable if T_1 and T_2 also are. So we regard $U_{min}(2) = 0.828 = 2(2^{1/2} - 1)$.

At the second scenario, T_1 has higher priority and makes its deadline. Let $T_1 = (414, 1000, 1000)$ and $T_2 = (c_2, d_2, 1414)$. We see that if $c_2 = 586$ and $d_2 = 1414$ the schedulability of T is guaranteed. Suppose that d_2 changes to $d_2 = 1400$. We notice that we cannot execute the first instance of T_2 after 1400 for example. Suppose that T_2 misses its deadline after 1400, finishing at 1401. This is possible if $c_1 = 400$ and $c_2 = 601$. Thus, $d_2 = 1400$ enables the existence of the utilization factor $U = 0.825$ that, although smaller than the minimum, does not guarantee the schedulability. In Figure 3 we see the T_1 and T_2 scheduling as above.

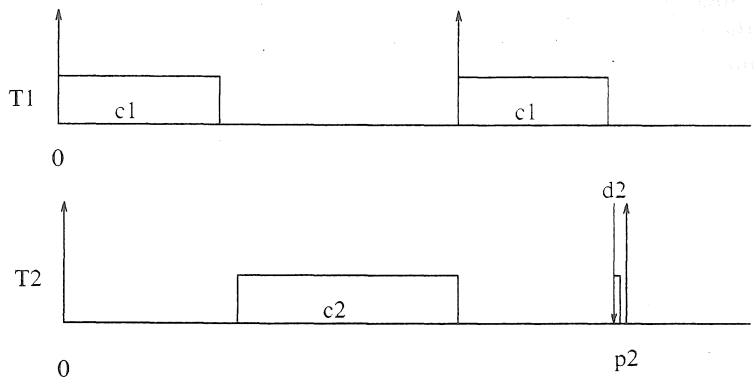

Figure 3: Missing an arbitrary deadline

Indeed, as seen in Figure 3, decreasing d_2 makes the utilization factor of T_1 decrease more than the increase of T_2 utilization, while fully using the CPU. This decreases the minimum utilization factor that enables the sufficient condition[5].

When T_1 and T_2 have arbitraries deadlines we have to take into account d_1, that may affect T_1 schedulability, as well as d_2 responsible for decreasing the minimum utilization factor.

3.2 Evaluation of the Minimum Utilization Factor

The minimum value for the utilization factor must be obtained according the approaches developed elsewhere, based on the minimum of utilization factor that fully utilizes the CPU[5]. In this sense, let us consider the system T as before, with $d_1 = p_1$ and $d_2 = lp_1 + q$, where l is a non-negative integer and $q < p_1$. We can show that, given such system, we can get T^* such that its utilization factor is smaller while keeping the CPU fully utilized.

Theorem 1 *Let $T = \{T_1, T_2\}$ be a system where $T_i = (c_i, d_i, p_i)$ with $d_2 = lp_1 + q$. If the system T^* given as $T = \{T_1^*, T_2^*\}$, where $T_1^* = T_1$ and $T_2^* = (c_2^*, p_1 + q, p_2^*)$, is schedulable T is also schedulable.*

Proof. In fact, we want to show that, if for a system with $d_2 < 2p_1$ the system is schedulable, then if $d_2 = lp_1 + q$ the new system will also be schedulable. Assume that $d_2 = p_1 + q$, where $q < p_1$, and $\{T_1, T_2\}$ are schedulable, we mean $U_{T^*} \leq U_{min}$. Now, change T_2 such that $d_2 = lp_1 + q$ and $p_2 = k_2 d_2$. If we keep the fully utilization up to the deadline, $c_2 = c_2^* + (l-1)(p_1 - c_1)$. So T_1 utilization will not change and T_2 utilization will depend on the new value of c_2. Originally either $c_2 = p_1 - c_1$ or $c_2 = d_2 - 2c_1$. If $c_2 = p_1 - c_1$ then $c_2 = lc_2^*$, and as $p_2 = lp_2^* - (l-1)k_2 q$, the minimum utilization factor that fully utilizes the CPU occurs when $d_2 < 2p_1$. On the other hand, if $c_2 = d_2 - 2c_1$ again we have a minimum utilization factor when $d_2 < 2p_1$. We may also substitute T_1 for $T_1^* = (c_1, ld_1, lp_1)$, meaning that the new period(and deadline) of T_1^* is l times greater. Thus $c_2^* = c_2 + (l-1)c_1$ in order to fully utilize the CPU. So, $U_{T^*} = c_1^*/lp_1 + c_2^*/p_2$, or $U_{T^*} = c_1^*/lp_1 + (c_2 + (l-1)c_1/p_2$. Then $U_{T^*} = c_1/lp_1 + (l-1)c_1/p_2 + c_2/p_2 \leq c_1/lp_1 + (l-1)c_1/lp_1 + c_2/p_2 = c_1/p_1 + c_2/p_2 = U_T$. Concluding, the smaller value for the fully utilization occurs when $d_2 < 2p_1$.

Any further analysis will use this result. Let us relax the above conditions, observing that, indeed, we may have a schedulable set even if $d_1 \leq p_1$. Let $U_{min}(2)$ be the minimum U value that guarantees the two-task set schedulability.

Theorem 2 *Let T be as above, such that $d_2 < 2p_1$, and $U_{min}(2)$ the minimum value of U to guarantee T_2 schedulability. The set T is schedulable if $d_1 \geq U_{min}(2)p_1$.*

Proof. As T_2 is schedulable anyway, only T_1 could miss its deadline and it would happen only if $c_1 > d_1$, implying a value of U such that $U > U_{min}(2)$. Thus, the set is schedulable if $U \leq U_{min}(2)$.

Let us now start the evaluation of the utilization factor minimum value. Initially we make an analysis based on three possibilities:

- $p_1 \leq d_2$.
- $d_2 \leq p_1 \leq p_2$.
- $p_2 \leq p_1$.

3.3 Common Scenarios

Let us evaluate the minimum utilization factor when $p_1 \leq d_2$. We set the following parameters:
$c_1 = m_1 d_1$
$p_1 = k_1 d_1 = d_1$
$d_2 = K d_1$, provided that $d_2 < 2p_1$ as stated in theorem 3.3.1. Yet, $p_2 = k_2 d_2 = K k_2 d_1$

We have three possibilities for c_1:

- $c_1 < d_2 - p_1$
- $c_1 > d_2 - p_1$
- $c_1 = d_2 - p_1$

Let us analyze each possibility in turn:

- $c_1 < d_2 - p_1$.

  The value of c_2 must be $d_2 - 2c_1$. Then we have:

  $c_1 = m_1 d_1$ e $c_2 = (K - 2m_1)d_1$

  Thus, $U = m_1/k_1 + (K - 2m_1)/Kk_2$, and rearranging:

  $U = 1/k_2 + m_1(Kk_2 - 2k_1)/Kk_1k_2$

  We realize that the U value increases with $c_1(m_1)$ if $Kk_2 - 2k_1$ is positive.

- $c_1 > d_2 - p_1$.

  We have $c_2 = p_1 - c_1$. Then $c_1 = m_1 d_1$ e $c_2 = (k_1 - m_1)d_1$, generating:

  $U = m_1/k_1 + (k_1 - m_1)Kk_2$

  $U = K/k_1 + (2k_1/k_2)(1/K) - (k_2 + 1)/k_2$

  As $Kk_2 - k_1 > 0$ this choice makes U increase with c_1. So we have to decrease c_1. Then the optimal value of c_1 is given by $c_1 = d_2 - p_1$. In this case $c_2 = (2k_1 - K)d_1$.

$U = m_1/k_1 + (2k_1 - K)/Kk_2$
$U = K/k_1 + (2k_1/k_2)(1/K) - (k_2 + 1)/k_2$

The following observations are valid when $p_1 \leq p_2 \leq 2p_1$. Our hypothesis is that $p_2 \leq 2p_1$. Further, we analyze what happens if $p_2 > 2p_1$.

The minimum value of U depends on K, given k_1 and k_2. As we assume $k_1 = 1$, K defines the optimal value for the ratio d_2/p_1 when there is a squeeze factor for T_2. The minimum value of U can be given deriving U with respect to K:

$dU/dK = 1/k_1 - 2(k_1/k_2)(1/K)^2 = 0$

We get the optimal value of $K = k_1(2/k_2)^{1/2}$. For $k_1 = 1$, we have $K = (2/k_2)^{1/2}$. The value for U_{min} is:

$U_{min} = (1/k_2)(2(2k_2)^{1/2} - (k_2 + 1))$

We observe that if $k_2 = 1$, we have $U_{min} = 2(2^{1/2} - 1)$

We now turn back to the case where $Kk_2 - 2k_1 > 0$, with $p_2 > 2p_1$. The minimum value is still valid provided $k_2 \leq 2$. So, we are assuming that $k_2 \leq 2$. Elsewhere we analyze the results for any k_2.

Let us take the case where k_1 is general. The change that must be made is related to what utilization factor we have to deal with. A very high k_1 value may minimize the utilization factor, given as $1/k_1$. As we are assuming that $c_1 \leq d_1$, this will pose no problem, which is not true in the general case. For this case, given k_1 e k_2, we try to guarantee that $d_1 \geq U_{min}(1)p_1$. Regarding T_2, it is also needed to guarantee that $d_2 \geq U_{min}(2)p_2$. This is necessary to accept $U_{min}(i)$ as the minimum utilization factor that guarantees tasks whose utilization factor are smaller.

We conclude our evaluation for the scenarios analized with the following observations:

- T_1 is schedulable if $U_1 \leq U_{min}(1) = 1/k_1$, as given before for U_{min}.
- T_2 is schedulable if $U_1 + U_2 \leq U_{min}(2)$.
- The above conditions are suitable if $k_2 \leq 2$. Otherwise $U_{min}(2) = 1/k_2$.

3.4 Uncommon Scenarios

Let us consider two uncommon scenarios:

- $d_2 \leq p_1 \leq p_2$
- $p_2 \leq p_1$

In the former case we point out the meaning of the preemptive algorithms, where higher priority tasks preempts lower ones. As $p_1 \geq d_2$ and T_1 has higher priority no preemption occurs. Thus, up to d_2 we execute only one instance of each task. Thus $c_1 + c_2 \leq d_2$, or equivalently, $c_2 = d_2 - c_1$. So $U = c_1/p_1 + (d_2 - c1)/p_2$, or $U \geq c_1/p_1 + (d_2 - c_1)/p_2$, since $p_1 \leq p_2$. Then $U \geq c_1/p_1 - c_1/p_2 + 1/k_2$. In this way, the minimum utilization factor occurs when $c_1 = 0$, or $U = 1/k_2$. So, we can understand the minimum utilization as being $U_{min}(1,2) = 1((2/k_2)^1 - 1) + (1 - 1/k_2)$, or $U_{min}(1,2) = 1/k_2$.

In the latter $c_2 = d_2 - c_1$, leading to $U = c_1/p_1 + (d_2 - c_1)/p_2$ and $U \leq c_1/p_1 + (d_2 - c_1)p_1$ and then c_1 should be a maximum in order that U be a minumum. This happens when $c_1 = d_1$ of $U_{min}(2) = 1/k_1 + (K - 1)k_2K$.

4 Schedulability Conditions for a n-task system

In the general case, we have an arbitrary periodic task system, each task with an arbitrary deadline, provided that if and only if $d_i \leq d_j$ then $p_i \leq p_j$. Let the n-task set be $T = \{T_1, \ldots, T_n\}$, such that $d_1 < d_2 < d_3 < \ldots < d_n$ e $p_i = k_i d_i$ for all $i = 1 \ldots n$. Also, as cited above, $p_1 < p_2 < \ldots, p_n$. As the Deadline Monotonic Algorithm is optimal, let us use it in our evaluations. Two cases must be observed:

- $d_i \leq p_i < d_{i+1}$ for $i = 1 \ldots n-1$ and $d_n \leq p_n$.
- $p_i \leq p_j$ for $i < j$.

Both cases are very representative of applications and will be analyzed.

4.1 Period between adjacent deadlines

First, we consider $d_i \leq p_i \leq d_{i+1}$, provided $d_n < 2p_1$. This last equation will give us the best relation for the Utilization Factor.

$U = \sum_{i=1}^{n} c_i/p_i$

In order to get the minimum from U fully using the CPU, we have to choose the execution times such that a small change in these values will not decrease U while fully using the CPU. Such values are:

$c_i = p_{i+1} - p_i$ for $i = 1 \ldots n-2$

$c_{n-1} = d_n - p_{n-1}$

$c_n = d_n - 2(c_1 + \ldots + c_{n-1})$

Thus, we get U given as:

$U = \sum_{i=1}^{n-2}(p_{i+1} - p_i)/k_i d_i + (d_n - p_{n-1})/k_{n-1}d_{n-1} + (d_n - 2(c_1 + \ldots + c_{n-1}))/k_n d_n$

Naming $k_{i,j} = d_i/d_j$, we have:

$U = k_2 k_{2,1}/k_1 + k_3 k_{3,1}/k_2 k_{2,1} + \ldots + k_{n,1}/k_{n-1}k_{n-1,1} + (1-n) + (2k_1 - k_{n,1})/k_n k_{n,1}$

As $\partial U/\partial k_{i,1} = 0$, we have:

$U = n((2/k_n)^{1/n} - ((n-1)k_n + 1)/nk_n)$

4.2 Ordered Periods

Let us consider a system in which the period task order is equivalent to the deadline task order, such that if $d_i \leq d_j$, then $p_i \leq p_j$. Let K be the first index such that $p_k \geq d_n$. We will see that any execution that come from the tasks T_k to T_{n-1} will increase the utilization factor.

Theorem 3 *Let T be given as above. If U_1 is given through T such that $c_k > 0$, for $p_k \geq d_n$, then for U_2 given by $c_k = 0$ and $c_n = c_n + c_k$, we have $U_2 \leq U_1$.*

Proof. As $p_k \geq d_n$, we have only one execution of T_k in d_n. As $p_n \geq p_k$ by hypothesis, the utilization factor contribution from T_k outperforms the one from T_n, if the execution time of T_k can be changed to the one from T_n and vice-versa. Thus, every computation from T_k must be null, in order to minimize the utilization factor. As it holds for every K, all the tasks with $p_k \geq d_n$ do not participate on the evaluation of the minimum utilization factor.

Regarding the system T as above the minimum utilization factor should take into account only tasks whose periods are smaller than the maximum deadline(d_n). For these tasks the condition behaviour will remain as it was in the last section. The minimum factor will be available if $c_i = p_{i+1} - p_i$, for $i < k$ and $c_{k-1} = d_n - p_{k-1}$. The c_n value is as before. The system behaviour is much like the same, excluding the $n-k+1$ tasks. So the minimum factor will be

$$U = l((2/k_n)^{1/l} - ((l-1)k_n + 1)/lk_n)$$

The schedulability analysis will apply whenever the order remains, for instance, if $d_i \leq d_j$ then $p_i \leq p_j$. Let it be a task set T of n tasks that satisfies this requisite. The question is to fullfill the schedulability for this task set, given U_{min} values. We define $U_{min}(a,b) = a((2/k_b)^{1/a} - ((a-1)k_b + 1)/ak_b)$. We analyze the schedulability of each task separately:

- T_1 is schedulable if $U(1) \leq U_{min}(1,1)$
- T_2 is schedulable if $U(2) \leq U_{min}(2,2)$ for the case where $d_2 \leq p_1$ and $U(2) \leq U_{min}(1,2)$ for the case where $d_2 \geq p_1$
- T_j is schedulable if $U(j) \leq U_{min}(i+1, j)$, where i is the index such that $p_i \leq d_j \leq p_{i+1}$.

This procedure covers all the possible cases and it has polynomial time complexity, as it uses a sort and search for each task.

5 Comparison among the Methodologies

Here we compare the methodologies proposed with the related work, mainly concerning computational cost and performance values, taken as a measure of the CPU utilization that can be achieved. We present a separate analysis for each one of the related methodologies comparing them with our proposal. We would like to emphasize that HRTS requires that the computational cost be predictably minimum in order to be applied at execution time.

5.1 Blocking Methodology

The blocking approach assumes that the task deadline may be smaller than the task period through the blocking factor, regarded as the time interval within the task period that the task will have to be blocked. This test imposes a polynomial computational cost, but its performance is poor, when compared with the Arbitrary Deadline Variation Approach. We show below that ADVA has, in the general case, a better efficiency, once any task set schedulable using the blocking is proved to be schedulable using ADVA.

Theorem 4 *If a set τ is schedulable using the blocking approach, then it is also schedulable using ADVA.*

Proof. Consider τ a periodic task set with $k_i = p_i/d_i$. Let $K = max(k_i)$. Thus $max(1-1/k_i) = 1-1/K$. As $B_i/p_i = 1-1/k_i$, and assuming τ is schedulable using the blocking approach, we have $U \leq n(2^{1/n}-1)-(1-1/K)$[12]. It is known that $1/k_n \geq 1/K$, implying $n((2/k_n)^{1/n}-1)+(1-1/k_n) \geq n((2/K)^{1/n}-1)+(1-1/K)$. On the other hand $n((2/K)^{1/n}-1)+(1-1/K) \geq n(2^{1/n}-1)-(1-1/K)$. Thus $U \leq n((2/k_n)^{1/n}-1)+(1-1/k_n)$ which implies that τ is schedulable using ADVA.

To see that ADVA is more general let τ be $\{T_1, T_2\}$, where $T_i = (c_i, d_i, p_i)$. If $1/k_1 = 0.8$ and $1/k_2 = 0.7$. Then τ is schedulable using ADVA if $U \leq 0.66$. Taking $T_1 = (20, 40, 50)$ and $T_2 = (17, 49, 70)$. Thus $U = 0.642$. However τ is not schedulable using the blocking approach.

5.2 Uniform Deadline Variation Approach

Regarding the computational cost, the Uniform Deadline Variation Approach is equivalent to ADVA. Both base their tests on the CPU utilization factor. The performance of the UDVA algorithm is at best equal to the one obtained using ADVA. This happens naturally since the ADVA algorithm is a generalization of the UDVA.

Theorem 5 *If a set τ is schedulable using UDVA then it is also schedulable using ADVA.*

Proof. Let T be a periodic task set such that $K = p_i/d_i$ for each task in T. As T is schedulable using UDVA, we will have $U \leq n((2/K)^{1/n}-1)+(1-1/K)$[9]. Thus, as $k_n = K$, $U \leq n((2/k_n)^{1/n}-1)+(1-1/k_n)$. Then the schedulability will also be valid using ADVA. Now take $k_i = p_i/d_i$ for each task T_i. Let T' the derived set from T, taking $k_i = K$, where $K = max(k_j)$ for all tasks. Thus, the task deadlines of tasks in T' are smaller than the ones from T. Thus, if T' schedulable, so is T. As T' is schedulable if $U \leq n((2/K)^{1/n} - 1) + (1 - 1/K)$. Since $k_n \leq K$ T will be schedulable using ADVA.

We notice that there is a task set schedulable using ADVA, but whose schedulability is not guaranteed using UDVA, as observed in the example below.

Let a two task set such that $p_1 = k_1 d_1$ and $p_2 = k_2 d_2$. Assume $k_1 = 0.9$ and $k_2 = 0.95$. So $k_1 > k_2$. This set is schedulable using ADVA if $U \leq 0.807$. However UDVA is supposed to also build a successfully schedule if $U \leq 0.783$. If $c_1 = 20$, $p_1 = 50$, $c_2 = 28$ e $p_2 = 70$, $U = 0.8$, and this set is not guaranteed to be scheduled using UDVA.

6 Final Comments

The ADVA is particularly interesting when the environment is complex and highly dynamic, and requires a very supportive scheduling algorithm. If changing the task set is a rule rather than an exception the interference analysis may no longer be useful, as an on-line approach.

Our ADVA methodology is very general, being restrictive only at rare situations. In any case, we observe that when the deadlines are arbitraly small, neither sufficient nor exact tests may give good performance results. In this case, variable priority association should outperform any fixed priority association.

As an avenue to pursue we are considering the inclusion of a ready time for each task, defined as the time instant the task may begin execution. This model may fit the inclusion of E/S in the processing models of HRTS.

References

[1] Vieira,S.L. and Magalhães,M.F.: 'On-line Sporadic Task Scheduling in Hard Real-Time Systems'. To be published in *International Journal of Computer Systems and Engineering*, 1997.

[2] Vieira,S.L.: 'Escalonamento Dinamico de Tarefas Periódicas e Esporádicas', Ph.D. Thesis, 1994. Unicamp, Brasil.

[3] Leung,J. and Whitehead,J.: 'On the Complexity of Fixed-priority Scheduling of Periodic, Real-Time Tasks', *Performance Evaluation*, 1992, (2), pp. 237-250.

[4] Audsley,N.C., Burns,A., Richardson,M.F. and Wellings,A.J.: 'Deadline Monotonic Scheduling Theory'. *Proc. of the IFAC Workshop on Real-Time Programming*, 1992, pp. 55-60.

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

[6] Sha,L. and Goodenough,J.: 'Real-Time Scheduling Theory and Ada', *Computer*, 1990, **23**(4)pp. 53-62.

[7] Lehoczky,J. and Sha,L.: 'Performance of Real-Time Bus Scheduling Algorithms'. *ACM Performance Evaluation Review*, 1986, (14).

[8] Audsley,N.: 'Deadline Monotonic Scheduling', Technical Report YCS 146, University of York, 1990.

[9] Barth, I.: 'Extending the Rate-Monotonic Scheduling Algorithm to Get Shorter Delays'. *Proc. of the 5th NOSSDAV*, 1994.

[10] Steinmetz, R.: 'Multimedia Technoloty', Springer-Verlag, 1993.

[11] Lehoczky,J.P.: 'Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines'. *Proc. IEEE 11st Real-Time System Symposium*, December 1990, pp. 210-209.

[12] Sha,L, Rajkumar,R. and Lehoczky,J.: 'Priority Inheritance Protocols: An Approach to Real-Time Synchronization'. *IEEE Trans. on Computer*, 1990, **39** pp. 1175-1185.