Ideas to Improve the Performance in Feasibility Testing for EDF

Alejandro Masrur Georg Färber
Institute for Real-Time Computer Systems
Technische Universität München, Germany
{Alejandro.Masrur, Georg.Faerber}@rcs.ei.tum.de

Abstract

In this paper, we prove analytically that it is possible to improve the feasibility bound presented by Ripoll et al. in [5] for the case where deadlines are less than or equal to periods. The improvement with respect to Ripoll's bound consists in a factor that depends on the processor utilization: the higher the processor utilization, the more significant the improvement. Although this factor may be negligible for some task configurations—particularly for low processor utilizations, we show by means of an example that for some other task configurations, it does improve the performance in feasibility testing for EDF.

1. Introduction

In [4], Liu and Layland showed that a feasibility test for synchronous tasks scheduled under EDF can be performed in polynomial time, when deadline (d_i) is equal to period (p_i) for all tasks. Posteriorly in [2], Baruah et al. proved that this is also the case when $d_i \geq p_i$ holds for all possible i.

When deadlines are allowed to be less than periods, the complexity grows considerably. However, assuming the processor utilization (U) to be less than 100%, Baruah et al. also proved that if a deadline is missed, this happens before a maximum time upper bound known as *feasibility bound*. This result allowed Baruah et al. to design a pseudo-polynomial time algorithm for the case where deadlines are not forced to be equal to periods.

Another pseudo-polynomial time algorithm for $d_i \leq p_i$ was presented by Ripoll et al. in [5]. Ripoll et al. presented two better feasibility bounds, which they combined in an efficient algorithm. The first of them, referred as Ripoll's bound in this paper, has following expression: $\frac{\sum_{i=1}^{n}(1-\frac{d_i}{p_i})\cdot e_i}{1-U}$, where e_i is the execution demand of the i-th task. It should be mentioned that this polynomial-time feasibility bound was independently obtained by George et al. [3], who also considered the case $d_i > p_i$. The other feasibility bound presented by Ripoll et al. is based on the busy period analysis, whose calculation itself presents pseudo-polynomial complexity. This latter pseudo-polynomial feasibility bound was also independently obtained by Spuri [6].

The existing algorithms, including the one of Albers and Slomka [1], enable an efficient feasibility testing for most practical applications. Nevertheless, it remains interesting to research into possibilities of improving the performance in feasibility testing. A better performance might be useful for those real-time systems where tasks are to be dynamically accepted or mapped to processors.

In the following sections, we formally define the inverse synchronous case and prove it equivalent to the well-known synchronous case. On the base of the inverse synchronization of tasks, we show that Ripoll's bound can be improved in a factor equal to $\frac{1}{1-U}$. This improvement is specially significant for very high processor utilizations, however, as it will be shown by means of an example, considering it might be beneficial for practical processor utilization too.

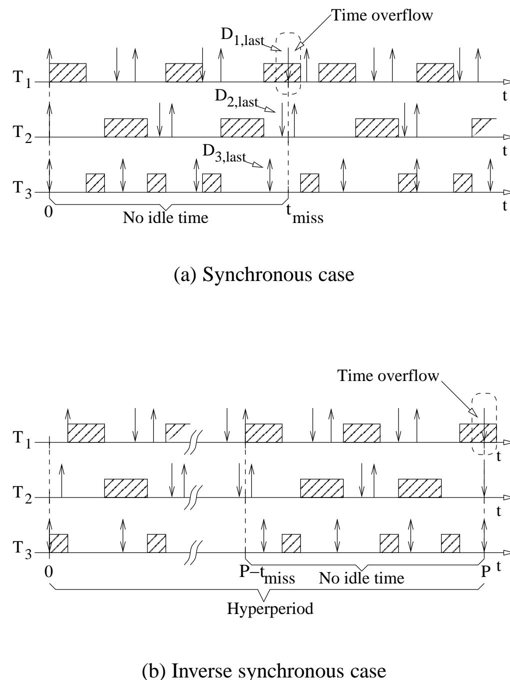
2. Task model and notation

We consider a set τ of periodic real-time tasks, which are fully preemptable and independent. Each task T_i is characterized by its period of repetition p_i, its relative deadline d_i and its worst-case execution demand e_i. Another parameter of tasks is the initial release time or phase ϕ_i. If $\phi_i = 0$ holds for all tasks in τ, the task set is called synchronous. On the other hand, if any phase in τ is not zero, the task set will be asynchronous.

In principle, a task T_i is an infinite succession of related jobs $J_{i,l}$, which have common execution demand and relative deadline. Additionally, each job has a univocal absolute deadline $D_{i,l}$. In this paper as in [5], relative deadlines are forced not to be greater than the respective periods, $d_i \leq p_i$ for all $i \leq n$, where n is the number of tasks in τ.

The hyperperiod P of the task set is equal to the least common multiple of all tasks' periods and the processor utilization is given by $U = \sum_{i=1}^{n} u_i$, where $u_i = \frac{e_i}{p_i}$ is the processor demand of T_i.

3. The inverse synchronous case



(a) Synchronous case

(b) Inverse synchronous case

Figure 1. Deadline miss

DEFINITION 1 *Given a set τ of periodic real-time tasks, the inverse synchronous case is such, where for each task in τ, the absolute deadline of the last job within the hyperperiod P coincide at the end of the hyperperiod, i.e. at time $t = P$.*

Unlike the synchronous case, where all first jobs are released together at the beginning of the schedule $t = 0$, in the inverse synchronous case, the absolute deadlines of all last jobs within the hyperperiod coincide for each task at $t = P$. In order to force the inverse synchronization of a task set τ of n periodic real-time tasks, the phase of each task $T_i \in \tau$ must be set to the value $\phi_i = p_i - d_i$ for all $i \leq n$. A task set whose tasks' phases are set according to this will be referred as *inverse synchronous task set*.

LEMMA 1 *Given a task set τ, whose $U \leq 1$, at least one of the last jobs in the inverse synchronous schedule misses its deadline at $t = P$, iif the synchronous schedule is not feasible.*

Proof: If $d_i < p_i$ for any task T_i in τ, not all phases will be equal to zero in the inverse synchronous case. So, the inverse synchronous case can be seen as an asynchronous schedule of the task set. Consequently, if the synchronous schedule is feasible, the inverse synchronous schedule will also be feasible—see [7]. For this reason, we concentrate in proving that if the synchronous schedule is not feasible, the inverse synchronous schedule is neither feasible and one of the last jobs misses its deadline at $t = P$.

We know from [4] that if a synchronous periodic task set is not feasible under EDF, then a deadline is missed at a time t_{miss} without idle time prior to it. Figure 1(a) illustrates this latter situation—upgoing arrows indicate release times whereas downgoing arrows indicate deadlines.

Now, we denote by $D_{i,last}$ the last absolute deadline of any T_i within $[0, t_{miss}]$. If all tasks are shifted right so that all $D_{i,last}$ coincide at t_{miss}, all jobs with absolute deadline less than or equal to t_{miss} remain completely within $[0, t_{miss}]$. By shifting tasks in this way, we have not changed the amount of processor workload in $[0, t_{miss}]$. As a consequence, a deadline is still missed at t_{miss}.

If the synchronous schedule begins at time $t = P - t_{miss}$ instead of $t = 0$, a deadline will be missed at $t = P$. Proceeding analogously to make all $D_{i,last}$ coincide this time at $t = P$, we can conclude that a deadline continues to be missed at $t = P$.

On the other hand, the inverse synchronous case forces the absolute deadlines of the last jobs within the hyperperiod to coincide at $t = P$. As all task in τ are periodic, the inverse synchronous case contains the schedule of the previously described situation in $[P - t_{miss}, P]$ and the continuation of it towards the origin in $[0, P - t_{miss})$, see figure 1(b).

For the case $d_i \leq p_i$, if no deadline is missed in the interval $[0, P]$ of the synchronous case, no deadline will be missed at all, i.e. $t_{miss} \leq P$—see [5]. Consequently, the inverse synchronous case will contain in the interval $[P - t_{miss}, P]$ the same workload as the synchronous case in the interval $[0, t_{miss}]$. The thesis follows. $\square$

COROLLARY 1 *If a synchronous τ is not feasible under EDF, there is a point in time $t_s = P - t_{miss}$ for the inverse synchronous case, from which the processor never idles till a deadline is missed at $t = P$.*

4. Improving Ripoll's feasibility bound

If a synchronous τ is infeasible, the inverse synchronous τ is neither feasible. In this latter case, the processor does not idle from t_s—defined in corollary 1—till a deadline is missed at $t = P$.

LEMMA 2 *Suppose that a new task set τ_f is obtained by adding a fictitious task T_f to τ, whose parameters are $p_f = d_f = P$, and $e_f = (1 - U) \cdot P$. τ_f is feasible under EDF, iif the initial task set τ is also feasible.*

Proof: The execution demand of T_f is equal to the total idle time in $[0, P]$ when scheduling τ. Additionally, there is only one job of T_f per hyperperiod because $p_f = P$. That is, the processor has enough idle time within $[0, P]$ to execute this only job of T_f before its deadline at $t = P$. We conclude T_f cannot affect the feasibility of τ. □

THEOREM 1 *If a synchronous τ is not feasible under EDF, the processor idles within the interval $[0, P]$ when scheduling the inverse synchronous τ_f of lemma 2.*

Proof: As the total utilization of τ_f is $U_f = U + \frac{e_f}{p_f} = 1$, if no deadline is missed, the processor will be the whole time busy regardless of the synchronization case. According to lemma 1, if the synchronous case is not feasible, a deadline is missed at $t = P$ in the inverse synchronous case. Now, if a deadline is missed at $t = P$, an amount of idle time equal to the time overflow at $t = P$ must be originated in the interval $[0, P]$. This latter is because $U_f = 1$, which means that the workload generated by τ_f does not suffice to busy the processor the whole time in $[0, P]$ and simultaneously to produce a time overflow at $t = P$. Notice that the amount of time overflow is executed after $t = P$ and that the schedule begins at $t = 0$, so there is no pending backlog at $t = 0$ that could fill this idle time up. In conclusion, the processor idles in $[0, P]$ despite the presence of the fictitious task T_f. □

LEMMA 3 *If the processor idles in $[0, P]$ when scheduling the inverse synchronous τ_f defined in lemma 2, the idle time can neither occur before the first job $J_{f,1}$ of the fictitious T_f finishes executing nor after $t_s = P - t_{miss}$ from corollary 1.*

Proof: the proof of this lemma is immediate. No idle time can occur as long as $J_{f,1}$ (the only job of T_f in $[0, P]$) has not finished executing, because $J_{f,1}$ would otherwise keep the processor busy. Additionally, as the processor never idles from t_s till a deadline is missed at $t = P$, the idle time can neither occur after t_s. □

LEMMA 4 *If all task parameters are integers, and $e_f = (1-U) \cdot P + 1$, $J_{f,1}$ finishes executing before $t_s = P - t_{miss}$ defined in corollary 1.*

Proof: If all task parameters are integers, the minimum possible time overflow will be 1, i.e. the idle time will be at minimum also 1; see theorem 1. Consequently, augmenting the fictitious execution demand in 1 time unit fills this minimum possible idle time up and we get closer to the value of t_s, which follows the idle time; see lemma 3. □

We know that if a synchronous τ is infeasible under EDF, $J_{f,1}$, with an e_f defined in lemma 4, finishes executing before t_s in the inverse synchronous case. For this reason, finding the point at which $J_{f,1}$ finishes executing allows us to estimate the value of t_s. Recall that in the inverse synchronous case, the phase of any task T_i is set to $\phi_i = p_i - d_i$ for all $i \leq n$. Now, supposing that T_r represents any possible *real task* from τ, we assume the following: **(1)** $J_{f,1}$ finishes executing when a job of T_r gets ready. **(2)** A job of each other task gets ready simultaneously with this particular job of T_r.

Considering that k_i represents the number of jobs of any T_i till the fictitious $J_{f,1}$ finishes executing, we can mathematically formulate our two assumptions as follows:

$$\sum_{i=1, i \neq r}^{n} k_i \cdot e_i + k_r \cdot e_r + e_f = k_r \cdot p_r + \phi_r, \quad (1)$$

$$k_i = \frac{k_r \cdot p_r + \phi_r - \phi_i}{p_i}. \quad (2)$$

Replacing equation 2 in 1 we get the following expression of k_r:

$$k_r = \frac{e_f - \phi_r + \sum_{i=1, i \neq r}^{n} (\phi_r - \phi_i) \cdot u_i}{(1 - \sum_{i=1}^{n} u_i) \cdot p_r}. \quad (3)$$

According to our first assumption, the point in time at which $J_{f,1}$ finishes executing will be given by:

$$t_f = k_r \cdot p_r + \phi_r \leq t_s. \quad (4)$$

LEMMA 5 *Inequality 4 holds, even if our assumptions do not hold, i.e. our two assumptions are pessimistic.*

Proof: If our first assumption does not hold, $J_{f,1}$ finishes executing some time Δ_f before a job of T_r gets ready. The right member of equation 1 will be given by $k'_r \cdot p_r + \phi_r - \Delta_f$—a positive Δ_f can be discarded because it would imply that $J_{f,1}$ has still not finished executing. Now, if the second assumption does not hold, the numerator in equation 2 will be $k'_r \cdot p_r + \phi_r - \phi_i \pm \Delta_i$. The sign of Δ_i will be positive, if a job of T_i gets ready after the job of T_r otherwise it will be negative. With help of figure 2, we will analyze what happens when every Δ_i is negative, which would represent the most adverse configuration. This latter is because not considering all negative Δ_i, t_f in equation 4 could be greater than the actually finishing time of $J_{f,1}$. Consequently, we could commit an error in estimating t_s. Here again, $\Delta_i \leq \Delta_f$ must hold for all i, otherwise it would imply that $J_{f,1}$ has still not finished executing.

The expression for k'_r will be: $k'_r = \frac{e_f - \phi_r + \Delta_f + (\phi_r - \phi_1 - \Delta_1) \cdot u_1 + (\phi_r - \phi_2 - \Delta_2) \cdot u_2}{(1-U) \cdot p_r}$. So if our assumptions do not hold, the finishing time of $J_{f,1}$ will be given by: $t'_f = t_f + \frac{\Delta_f - \Delta_1 \cdot u_1 - \Delta_2 \cdot u_2}{1-U} - \Delta_f$.

Replacing Δ_2 and Δ_1 by Δ_f, we get: $t'_f > t_f + \frac{(U - u_1 - u_2) \cdot \Delta_f}{1-U}$. As $U > u_1 + u_2$ holds, $t_f < t'_f$ also holds, where t_f is given by equation 4. The thesis follows. □

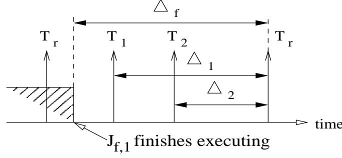


Figure 2. Our assumptions do not hold

To obtain a feasibility bound in terms of the *normal* synchronous case, we use the fact that $t_s = P - t_{miss}$. So, we can estimate t_{miss} in the following way: $t_{miss} = P - t_s \leq P - t_f$, where t_f is given by equation 4. So, replacing e_f for the value $(1-U) \cdot P + 1$ in equation 3 and proceeding as mentioned, we obtain: $t_{miss} \leq P - \frac{(1-U)\cdot P + 1 - \phi_r + \sum_{i=1, i\neq r}^{n} (\phi_r - \phi_i)\cdot u_i}{1-U} - \phi_r$. Recalling that $\phi_i = p_i - d_i$, we finally get:

$$t_{miss} \leq \frac{\sum_{i=1}^{n} (1 - \frac{d_i}{p_i}) \cdot e_i}{1-U} - \frac{1}{1-U}. \qquad (5)$$

Clearly, the factor $\frac{1}{1-U}$ will improve the performance in feasibility testing for EDF only if the analyzed task set is feasible. Otherwise, a time overflow will always be found before the time upper bound given by equation 5.

Although the factor $\frac{1}{1-U}$ might be negligible for low processor utilizations, there are practical cases for which it is not. The example for $U \sim 88\%$ illustrated in figure 3 should make this latter clear. In figure 3, Ripoll's bound and the busy-period feasibility bound are indicated by I and I_{busy} respectively. For this example, considering the factor $\frac{1}{1-U}$ makes possible to avoid the verification of two deadlines.

Finally, let us consider the following task set: $p_1 = 75$, $d_1 = 70$, $e_1 = 15$, $p_2 = d_2 = 668$, $e_2 = 334$, $p_3 = 180$, $d_3 = 178$ and $e_3 = 54$. This task set presents a total processor utilization of 100%, however, it is still feasible. As $U = 1$ holds for this example, it will be impossible to calculate equation 5, since denominators will be zero. Consequently, we will not be able to provide an upper bound for the feasibility test and all deadlines within $[0, P]$ must be verified [5]. Now, if we consider that $e_2 = 333$ holds in the previous example, $U < 1$ will also hold. For this latter case, Ripoll's bound will be $I \sim 1069$ while $I - \frac{1}{1-U} \sim 401$, which is less than a half of Ripoll's bound.

5. Conclusions

In this paper, we proved that Ripoll's feasibility bound for the case $d_i \leq p_i$ can be improved in a factor equal to $\frac{1}{1-U}$. This factor grows hyperbolically as the processor utilization nears 100%, which makes its consideration especially convenient in case of high processor utilizations. As task sets with $U = 100\%$ can theoretically be feasible, task

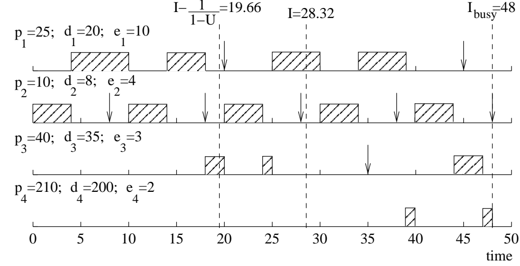


Figure 3. Example for $U \sim 88\%$

sets with processor utilization infinitesimally close to 100% can also be feasible. For these theoretical cases, the improved feasibility bound should be considered.

Additionally, we showed an example with $U \sim 88\%$ for which the improved feasibility bound presents little but not negligible advantage with respect to the others. When the processor utilization tends to 0, the factor $\frac{1}{1-U}$ tends to 1. However, its calculation does not present much more computation cost and might still be advantageous.

It should be noticed that equation 5 was obtained on the base of lemma 4, which assumes that all task parameters are integers. If real numbers are considered, the improvement proposed in this paper will be given the factor $\frac{10^{-k}}{1-U}$, where k is the maximum number of decimal places allowed. All conclusions taken before are still valid. For example, consider the task set derived from the previous one: $p_1 = 7.5$, $d_1 = 7$, $e_1 = 1.5$, $p_2 = d_2 = 66.8$, $e_2 = 33.3$, $p_3 = 18$, $d_3 = 17.8$ and $e_3 = 5.4$. For this case, Ripoll's bound will be $I \sim 106.9$ while $I - \frac{0.1}{1-U} \sim 40.1$, which is still less than a half of Ripoll's bound.

References

[1] K. Albers and F. Slomka. Effi cient feasibility analysis for real-time systems with edf scheduling. *Proceedings of the Date 05 Conference*, March 2005.

[2] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. *Proceedings of the Real-Time Systems Symposium*, December 1990.

[3] L. George, N. Rivierre, and M. Spuri. Preemptive and non-preemptive real-time uniprocessor scheduling. *Rapport de Recherche RR-2966, INRIA*, 1996.

[4] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in hard real-time environments. *Journal of the Association for Computing Machinery*, 20(1):40–61, 1973.

[5] A. Ripoll, I. Crespo and A. Mok. Improvement in feasibility testing for real-ttime tasks. *Real-Time Systems*, 11(1):19–39, Jul 1996.

[6] M. Spuri. *Earliest Deadline Scheduling in Real-Time Systems*. PhD Thesis at Scuola Superiore S. Anna, Italy, 1995.

[7] J. Stankovic, M. Spuri, K. Ramamritham, and G. Buttazzo. *Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms*. Kluwer, 1998.