Run-time efficient feasibility analysis of uni-processor systems with static priorities

Karsten Albers
Department for Embedded Systems/Real-Time Systems, University of Ulm
karsten.albers@informatik.uni-ulm.de

Frank Bodmann
Department for Embedded Systems/Real-Time Systems, University of Ulm
frank.bodmann@informatik.uni-ulm.de

Frank Slomka
Department for Embedded Systems/Real-Time Systems, University of Ulm
frank.slomka@informatik.uni-ulm.de

Abstract: The performance of feasibility tests is crucial in many applications. When using feasibility tests online only a limited amount of analysis time is available. Run-time efficiency is also needed for testing the feasibility of many different task sets, which is the case in system synthesis tools. We propose a fast uni-processor feasibility test using static priorities. The idea is to use approximation with a variable error to achieve a high performance exact test. It generally outperforms the existing tests which we show using a large number of random task sets.

Keywords: Real-Time Systems, Fixed-Priority Scheduling, Feasibility Analysis, FPTAS.

1 INTRODUCTION

The performance of the real-time analysis is important for the automation of the design process for embedded systems. Three different kinds of feasibility tests exist: Exact test, sufficient tests and approximations. The most used exact test is the worst case response time analysis by Audsley et al. (1993). Sjödin and Hansson (1998) propose an efficient implementation of this analysis. The exact test proposed by Manabe and Aoyagi (1998) has exponential time complexity and is outperformed by the worst case response time analysis in most cases. The sufficient test by Liu and Layland (1973) and by Bini et al. (2001) are designed for RM scheduling and therefore not suitable for more general models. Approximations allows a trade-off between run-time and acceptance rate. A predefined error allows a certain amount of inexactness

while performing the test which leads to a speed-up for the tests. Albers and Slomka (2004) gives such an approach for earliest deadline first (EDF) scheduling and extended it to a fast exact schedulability analysis (Albers, Slomka 2005). It takes advantage of the approximation approaches and allows to dynamically adjust the error during the schedulability analysis. Fisher and Baruah (2005a, 2005b) transferred this approximation to systems with static priorities. The goal of this paper is now to extends the results achieved of the exact dynamic test to systems with fixed priorities.

2 MODEL

We consider the sporadic task model on uniprocessor systems using preemptive fixed priority scheduling. The results can be extended to more advanced task models. Each task τ_i of a task set $\Gamma=\{\tau_1,..., \tau_n\}$ is described by an initial release time (or phase) ϕ_i, a relative deadline D_i (measured from the release time), a worst-case execution time C_i (cost) and a minimal distance (or period) T_i between two instances of a task.The priorities assignment of the tasks follows the deadline monotonic approach. The task wih the smallest deadline gets the highest priority. An invocation of a task is called a job, and the k^{th} invocation of each task τ_i is denoted $\tau_{i,k}$. The release time $\phi_{i,k}$ of $\tau_{i,k}$ can then be calculated by $\phi_{i,k} = \phi_i + (k-1) \cdot T_i$, and the deadline $d_{i,k}$ by $d_{i,k} = \phi_i + D_i + (k-1) \cdot T_i$. In the following the synchronous case is assumed, so $\forall \tau_i \in \Gamma : \phi_i = 0$. This leads also to a sufficient test for the asynchronous case. One feasibility test for this kind of task sets is proposed in by Baruah (2003) using the concept of demand and request bound functions. The idea is to define functions calculating the maximum sum of execution times of all jobs within an interval of length *I*. The request bound function considers all jobs that can arrive within a time interval of the length *I*. For the demand bound function it is additionally necessary that the deadline of all jobs occurs within *I*. To get the maximum demand bound contribution of a task τ_i it is necessary to calculate the sum of costs of all *k* consecutive jobs of the tasks for which the distance of the deadline of the *k*-th job of the task τ_i to the release time of the first job of the task τ_i is smaller than the length of the interval *I*: $d_{i,k} \le I$. *The maximum cumulated execution requirement of all jobs $\tau_{i,k}$ with $\tau_i \in \Gamma$ having request time and deadline within I* is called demand bound function:

$$dbf(I, \Gamma) = \sum_{\forall \tau_i \in \Gamma \wedge I \ge D_i} \left\lfloor \frac{I - D_i}{T_i} + 1 \right\rfloor \cdot C_i$$

Let Γ_τ be the task set that contains only task τ. We define $dbf(I, \tau) = dbf(I, \Gamma_\tau)$ as a shortcut description. Baruah also defines the request

bound function: *The maximum cumulated execution requirement of all jobs $\tau_{i,k}$ with $\tau_i \in \Gamma$ which have their release times within I*

$$\mathrm{rbf}(I, \Gamma) = \sum_{\forall \tau_i \in \Gamma} \left\lceil \frac{I}{T_i} \right\rceil \cdot C_i$$

Same as above we also uses $\mathrm{rbf}(I, \tau) = \mathrm{rbf}(I, \Gamma_\tau)$ as a shortcut description. For an interval I and a task τ_i a set of consecutive jobs of the task contributes fully to the request bound function of *I*, if the difference between the release time of the first job of the set and the release time of the last job of the set is equal or smaller than the length of *I*. For a feasibility test only intervals that start at an idle point of the system are of interest (i. e. a point in time where no request is ready for execution). The synchronous release of all tasks τ_i is taken as the start point of all intervals. For systems using EDF scheduling the following condition leads to a necessary and sufficient test: $\forall I > 0 : \mathrm{dbf}(I, \Gamma) \leq I$ Based on this condition Baruah (2003) gives a schedulability analysis and Albers and Slomka (2004) an approximation. For static priority scheduling, it is not sufficient to test the demand bound function. Instead it is necessary to consider for each task the interruption of higher priority tasks and therefore each task separately. Let $\Gamma_{hp(\tau)}$ be the task set which contains all task with a priority higher than that of task τ. A feasibility test for such a system is given by Baruah (2003): *A task system is feasible in regard to task τ if and only if for each absolute deadline of a job $d_{\tau,k}$, $k \in N$ there exist an interval $I' \leq d_{\tau,k}$ (and $I' \geq d_{\tau,k} - T_\tau$) for which the following condition holds:* $\mathrm{dbf}(d_{\tau,k}, \tau) + \mathrm{rbf}(I', \Gamma_{hp(\tau)}) \leq d_{\tau,k}$.

A job can satisfy its deadline if there exists an interval *I*, with length equal or smaller than the deadline of the task, in which the system has an idle point with respect only to the regarded jobs of this task and all jobs of higher priority tasks. One of these intervals is the response time of the task in question. Baruah proposes to first check the inequation above for $I' \leq d_{\tau,k}$ and than calculate the response time if necessary. To keep the test tractable the number of tested jobs of τ_i needs to be limited by an upper test border. Baruah has shown such a border for the recurring real-time task model.

3 APPROXIMATIONS

As a pre-step for achieving an exact test with a good performance, it is necessary to develop a new approximation for static priority scheduling. In contrary to the previously know approximation we need an algorithm in which the contributions of the tasks are calculated in an incremental way. To get such a selectable sufficiency the error which occurs due to the approximation needs to be bounded.

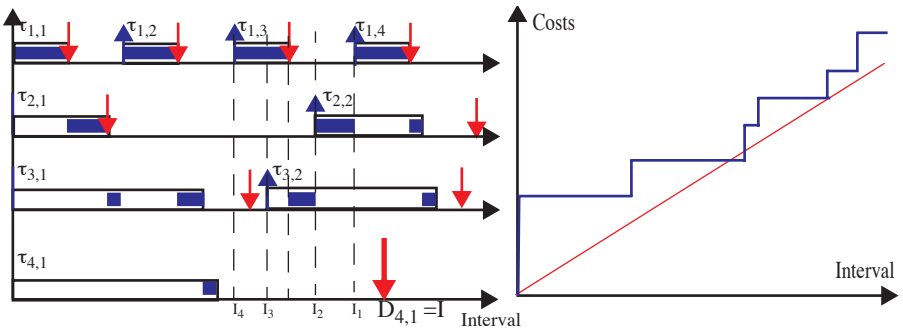

Figure 1. a) Schedule for example, b) Cumulated request for schedule

3.1 Static feasibility test

The idea of the approximation is to define a cumulated request bound function: *The cumulated request with respect to task τ is the request of all jobs of all tasks with a higher priority than τ which can arrive within an interval I and the demand of τ for this interval*:

$$\mathrm{Cumu}(I, \tau) = \mathrm{rbf}(I, \Gamma_{hp(\tau)}) + \mathrm{dbf}(I, \tau)$$

Let us now consider the cumulated request for those intervals for which the condition $I' = d_{next}(I', \tau)$ holds. *The additional request for interval I is the difference between the cumulated request and the interval:* $\mathrm{addReq}(\tau) = \mathrm{Cumu}(I, \tau) - I$. *The part of the jobs that cannot be processed within I due to the arrival times of the jobs is called exceeding costs.*

There are two possible situations in which exceeding costs can occur. In the first one the arrival of a job occures so late that the remaining time to the end of I is shorter than the execution time necessary for the job. It can than never be completed within I. Consider the example task set $\tau_1 := (C_1 = 4, D_1 = 4, T_1 = 8)$, $\tau_2 := (C_2 = 3, D_2 = 7, T_2 = 22)$, $\tau_3 := (C_3 = 3, D_3 = 17, T_3 = 19)$, $\tau_4 := (C_4 = 1, D_4 = 26, T_4 = 30)$. The schedule can be found in Fig 1a and the cumulated costs function belonging to it in Fig 1b. The job $\tau_{1,4}$ belongs to the first situation. The difference between its release time ($\varphi_{1,4}=24$) and I ($d_{4,1}=26$) is smaller than its execution time. One part of this job (length 2) has to be scheduled outside of *I* and therefore counts to the exceeding costs. The second kind of situation occurs when the task is preempted and therefore its execution is delayed so that at least a part of the execution has to occur after the end of *I*. One example for this situation is the job $\tau_{2,2}$ in Fig 1. Due to its preemption by $\tau_{1,4}$ it is not possible to process $\tau_{2,2}$ completely within *I*, therefore it partly counts to the exceeding costs. Same for $\tau_{3,2}$ which is delayed and preempted by three other jobs ($\tau_{1,3}$, $\tau_{2,2}$, $\tau_{1,4}$). All tasks with a priority higher than the priority of τ can contribute to the exceeding costs and it is not relevant to know which task exactly contributes how

much: Lemma: *The deadline of a job $\tau_{i,k}$ is met if and only if either for I = $d_{i,k}$: the cumulated request for I is smaller or equal to I (cumu(I,τ_i)≤I) or the additional request of I is covered by exceeding costs*. *Proof:* Suppose that the deadline of a job τ_i is not met despite that the cumulated request does not exceed the sum of *I* and the exceeding costs. Than costs of at least one job do not fit within *I* and the execceding costs. This is only possible if either there exists an idle time in I (with respect to τ_i and all tasks with a higher priority) or a job is processed within *I* that is not included in *cumu*. In case of the idle time $\tau_{i,k}$ has also been processed completly before the end of its deadline. The second case is that a job of a higher priority task, which has not been requested within *I* is processed within *I*. *I* starts at an idle point of the system (by definition) and processing a task with a request time outside of *I* would be in contradiction to the definition. The other possibility would be to process a job $\tau_{i,m}$ with *m>k*. But this can only be done if the considered job $\tau_{i,k}$ is already finished, so the deadline $d_{i,k}$ holds.

The idea for an implementation is to start with the additional request and try to reduce it step by step by considering the exceeding costs. If they cover the additional request the test succeeds otherwise the test fails.

3.2 Approximation with bounded deadlines

Using the proposed algorithm an approximation can be achieved. The idea is to limit the amount of test points for the cumulated request function. For each task only a limited amount of test points (one for each of the first *k* jobs) are considered exactly. The remaining test points are skipped and an approximation is used instead. In the example in Fig 2 the first two jobs of τ_1 are considered exactly, the remaing are approximated using the specific utilization of the task. *An upper bound for the task request bound function considering only the first k jobs exactly is called upper approximated task request bound function:*

$$rbf_{sup}'(I, \Gamma_i, k) = \begin{cases} rbf(k \cdot T_i, \Gamma_i) + \dfrac{C_i}{T_i} \cdot (I - d_{i,k}) & I > k \cdot T_i \\ rbf(I, \Gamma_i) & I \le k \cdot T_i \end{cases}$$

$$rbf_{sup}(I, \Gamma, k) = \sum_{\forall \tau_i \in \Gamma} rbf_{sup}'(I, \Gamma_i, k)$$

The error of the approximation is bounded as shown in Fig 2. The difference between the approximated and the real request bound function of task τ_i is bounded by one times C_i. In the approximated part of the function the value of rbf_{sup} is at least $k*C_i$. So, as rbf is a non decreasing function the relative error ε is limited to $1/k$.

$$\varepsilon = \frac{cumu_{sup}(I, \Gamma_i, k) - cumu(I, \Gamma_i)}{cumu(I, \Gamma_i)} \leq \frac{1}{k}$$

This also limits the overall error to *1/k*. *k* and therefore the degree of exactness can be chosen. The complexity of this approach can be calculated by considering that this test has to be done for each task separately. For a task τ_i with $D_i \leq T_i$ only the first job of the task has to be considered. The number of test points (of the approximated cumulated request bound function) is limited to k*n, where n is the number of tasks with a higher priority than τ_i. This leads to an overall complexity $O(log\ *n^2*1/\varepsilon)$.

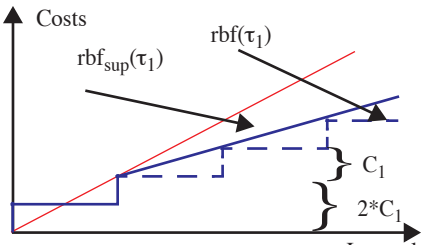

*Figure 2.*Approximated rbf_{sup} of τ_1

3.3 Approximation with arbitrary deadlines

The proposed approximation can be extended to the case of arbitrary deadlines ($D_i > T_i$). The question is which jobs of τ_i are necessary to be tested. For the periodic general task model, it was sufficient to test only the first job. For arbitrary task systems this is not sufficient any more because several jobs with the same priority can exist concurrently. Baruah (2003) has shown a border for the recurring real-time task system. The idea is to find the point of time where it is guaranteed that the cumulated request bound function does not exceed the intersection any more. *There exists a point of time $t_0 \geq 0$ such that for all $t' \geq t_0$: $cumu(t', \tau) \leq t'$*. Using the approximated cumulated request function such a point is guaranteed to exist using a task system with an overall utilization lower than 1. Starting at the point where the *rbf* for the last task switch to approximation, the remaining function is monotonically increasing with a slope equal to the utilization of the function. Therefore at one point this function must fall below the intersection. The overall amount of test intervals for each task is limited to *k*. This is the case even if several instances of a task are considered. Therefore the overall amount of test intervals is limited to *n*k* where n is the number of tasks. Therefore the complexity of this test is the same as in the non-general case.

4 EFFICIENT DYNAMIC TEST

To improve the performance of the exact test a combination of the exact test and an approximation is proposed. The idea is to use the approximation for skipping as many test points as possible.

The test is done using the request bound function. For the feasibility test of one task it is sufficient to show that, for each deadline of the tasks the request bound function for one test interval smaller than the deadline, meets or steps below the intersection. The problem is therefore to find this interval. An overestimating request bound function is not applicable to get an exact test. See Fig 4a for an example. If only the approximated request bound function is regarded the only acceptable test interval would be missed. So in contrast to the approximative test, it is not possible to use rbf_{sup}, the overestimating approximation function. Therefore an underestimating approximation (rbf_{inf}‘) is used: *A lower bound for the task request bound function considering only the first k jobs exactly.*

```
function ApproxSchedFunc(τ,Γh,I)
cumureq = Γh.cumu(I,τ)
if (cumureq ≤ I) ⇒ return true
∀τi ∈ Γh : testlist.add(τi, lastJobBefore(I,τi))
approxList() = {}
Uapprox =0
Itest= I
approxReq = cbfsup'(I, Γ)
while (Itest > I-Tτ)
   τi, Itest = testlist.getLast()
   approxReq = approxReq - Ci - Uapprox *(Itest - Iold)
   if (approxReq ≤ Itest)
        reduceApproximation (approxReq, testList, approxList)
        if (approxReq ≤ Itest)
             ⇒ return true
   if (τi ∈ approxList )
        approxList.remove(τ,)
        Uapprox = Uapprox - Ci/ Ti
   testlist.add(τi, Itest - Ti)
   Iold = Itest
end while
⇒ return false
end function
```

Figure 3. Dynamic Approximation

$$rbf_{inf}(I, \Gamma_i, k) = \begin{cases} rbf(k \cdot T_i, \Gamma_i) + \dfrac{C_i}{T_i} \cdot (I - d_{i,k}) - C_i & I > k \cdot T_i \\ rbf(I, \Gamma_i) & I \le k \cdot T_i \end{cases}$$

Consider Fig 4b. The approximation rbf_{inf} is comparable to the approximation rbf_{sup}. In contrary to rbf_{sup} it underestimates the request bound function. In case the test seems to succeed using this approximation it is therefore necessary to reduce the approximation step by step until the exact case is reached. Than the feasibility test with regard to interval I succeeds. It is only necessary to calculate the exceeding costs until they meet the additional

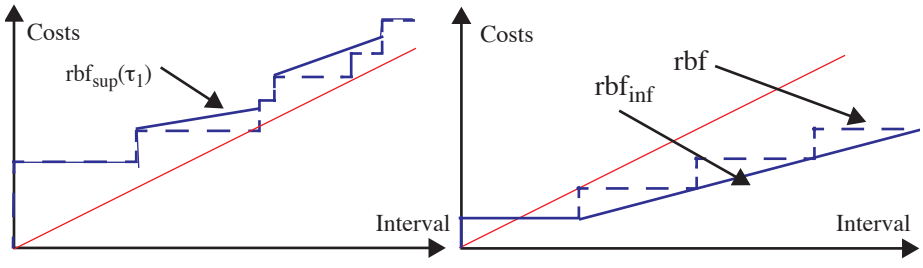

Figure 4. a) Approximated rbf_{sup} of τ_1 b) Approximated rbf_{inf} of τ_1

request. If this is the case, the deadline D_n is met. The algorithm are shown in Fig 3 and Fig 6. The idea of approximation as early as possible can be used for a further improvement of the test. It is necessary to calculate the cumulated costs at least once for each priority level. Evaluating in priority order, an approximative request bound function can be build step-by-step and used as a preliminary check. This allows to quickly check the feasibility for task with low priorities. The idea is to use the upper approximated request bound function for each task with no exact test interval. For any interval I this function can be derived: $rbf_{sup}(I, \tau, 0) = C_\tau + I \cdot C_\tau / T_\tau$

Only one part depends on the length of the interval. It is possible to calculate the sum of the execution times and the sum of the specific utilizations. This calculation can be done step-by-step during the process of testing the different levels of priorities. Evaluating the level in their priority order it is only necessary at each level to add the values for only one task. If the approximated cost exceeds the deadline, the approximated request bound functions are replaced with the exact values for each task step-by-step until either the costs doesn‘t exceed the interval any more or all approximations are revised.

```
function reduceApproximation(approxReq, testList, approxList)
  while (approxReq ≤ I_test )
    if (approxList = {})
      return
    τ_i= approxList.getNext()
    approxReq = approxReq + (⌈I_test / T_τ⌉ − I_test / T_τ) C_τ
    U_approx = U_approx - C_i/ T_i
  end while
⇒ return
end function
```

Figure 6. ReduceApproximation

5 EXPERIMENTS

We have tested the proposed algorithms using a large amount of randomly generated task graphs. We measured the run-time by counting the iterations needed by the algorithm to find a decision. For the new test we counted every iterations in the main loop, every reverse of an approximation and every iteration needed to get the initially values. The effort is compared to the effort of previous algorithms using the same task sets. For comparison we used an efficient implementation of the worst-case response time analysis introduced by Sjödin and Hansson (1998) and of the algorithm for analysis recurring real-time task sets proposed by Baruah (2003). We did not consider arbitrary tasks or more advanced task models here. Fig 7 a) shows an experiment which uses 50,000 randomly generated task sets with a utilization between 50% and 99%. The new algorithm shows an improved performance. It outperforms the algorithm by Baruah and only needs between 200 and 1,042 iterations in the average for task sets with different utilization instead of 4,950 to 8,574. The effort for the worst case response time analysis even with a very efficient

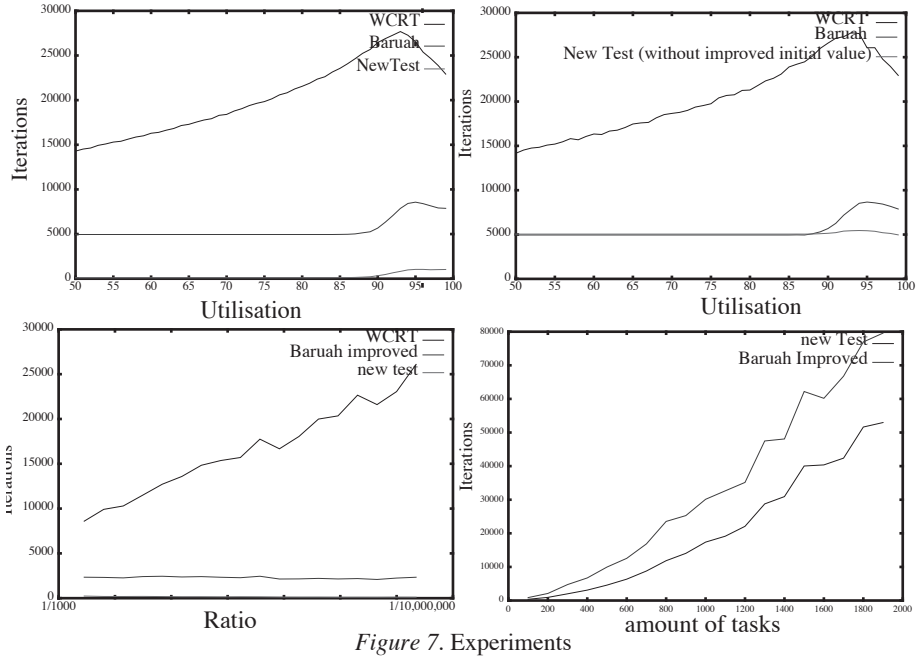

Figure 7. Experiments

implementation needs between 14,000 and 28,000 iterations and therefore much more than the other algorithms. In cases that the response time is not needed the new test delivers the result with less effort. To investigate how much of the improvement results only out of the new calculation of the initial value, Fig 7 b) shows an experiment (using 10,000 task sets) in which the new test does not use the improved approximative calculation of the initial values (Section 4). Despite that both algorithm needs a comparable effort for task sets with low utilization, the algorithm by Baruah needs up to 8,500 iterations whereas the new algorithm only needs up to 5,500 iterations. 5,000 iterations are needed alone for calculating initial values. To compare only the impact by the main algorithm an improved version of the test by Baruah was build for the following experiments. It uses also the improved calculation for the initial values. Fig 7 a) shows an experiment using different ratios between the smallest and the largest tasks in the task set. It is obvious that the effort for the worst-case response time analysis depends on the ratio between the largest and the smallest tasks in the task set. But the effort of the new test and the test by Baruah does not depend on the ratio. Indeed it declines a bit with a rising ratio due to a slightly higher acceptance rate of task sets with a high ratio. In Fig 7 b) the dependency between the effort and the amount of tasks in a task set is shown. To investigate the improvement due to the dynamic approximation both algorithm uses the improved cumulated cost calculation. The results shows that both algorithms depends on the amount of tasks in a comparable

way. Despite this the new algorithms seems to need less effort than the algorithm by Baruah in its improved implementation.

6 CONCLUSION

In this paper we proposed a new fast sufficient and necessary test for the feasibility analysis of preemptive static priority scheduling. Therefore a new approximation for static priority scheduling was also developed. The purpose of the approximation is to limit the run-time of the test. The new exact test, which also uses the idea of approximation, leads to a lower run-time for feasibility tests which also seems to be independent of the ratio between the largest and the smallest task.

References

K. Albers, F. Slomka. *An Event Stream Driven Approximation for the Analysis of Real-Time Systems*. Proceedings of the 16th Euromicro Conference on Real-Time Systems, Catania, 2004

K. Albers, F. Slomka. *Efficient Feasibility Analysis for Real-Time Systems with EDF Scheduling*, Proceedings of the Design Automation and Test in Europe Conference 2005 (DATE‘05), Munich, 2005

N.C. Audsley, A. Burns, M.F. Richardson, A.J. Wellings. *Hard real-time scheduling: The deadline monotonic approach*. Proceedings of the 8th Workshop on Real-Time Operating Systems and Software, 1993

S. Baruah *Dynamic- and Static-priority Scheduling of Recurring Real-Time Tasks*. Real-Time Systems. 24, 2003.

E. Bini, G. Buttazzo, G. Buttazzo. *The Hyperbolic Bound for Rate Monotonic Schedulability*. Proceedings of the Euromicro Conference on Real-Time Systems, 2001.

N. Fisher, S. Baruah. *A polynomial-time approximation scheme for feasibility analysis in static-priority systems with arbitrary relative deadlines*, Proceedings of the 17th Euromicro Conference on Real-Time Systems, Palma de Mallorca, 2005

N. Fisher, S. Baruah. *A Polynomial-Time Approximation Scheme for Feasibility Analysis in Static-Priority Systems with Bounded Relative Deadlines*, Proceedings of the 13th International Conference on Real-Time Systems, Paris, 2005

C. Liu, J. Layland. *Scheduling Algorithms for Multiprogramming in Hard Real-Time Environments*, Journal of the ACM, 20(1), 1973.

J.P. Lehoczky, L. Sha, J.K. Stronsnider, H. Tokuda. *Fixed Priority Scheduling Theory for Hard Real-Time Systems*. Foundation of Real-Time Computing: Scheduling and Resource Management, 1991

Y. Manabe, S. Aoyagi. *A Feasibility Decision Algorithm for Rate Monotonic and Deadline Monotonic Scheduling*, Real-Time Systems, 14, 1998

M. Sjödin, H. Hansson. *Improved Response-Time Analysis Calculations*, Proceedings of the RTSS, Madrid, Spain, 1998

-