Schedulability Analysis of Fixed Priority Systems with Scheduling Overheads

Yanfeng Zhai
School of Computer and Information Science
Southwest University
Beibei District
Chongqing, China

Fengxiang Zhang*
School of Computer and Information Science
Southwest University
Chongqing, China
*Corresponding author: zhangfx@swu.edu.cn

Abstract—In this paper, the author extends the traditional exact schedulability analysis for fixed priority (FP) preemptive scheduling by taking into account the extra scheduling overhead that may be induced by context switching. Then the author develops a necessary and sufficient scehdulability test for fixed priority scheduling on a single processor by considering the worst-case task response time. In the paper, the existing results on exact response time analysis have been discussed for fixed priority preemption schecduling with task set that is periodic or sporadic periodic respectively. As a generalization of fixed priority preemptive scheduling, the fixed priority preemption-threshold scheduling is also described in this paper. An improvement upon the previous results has been proposed by considering the influence of task response time caused by the context switching overhead and release jitter. Accounting for context switching overhead needs to increase the execution time of each task. The author also extends this analysis to consider the task response time with arbitrary deadlines.

Keywords-schedulability analysis; fixed priority; scheduling; response-time; scheduling overhead

I. INTRODUCTION

The schedulability analysis for real-time system is important because it can make sure whether the task set is schedulable or not, especially for hard real-time system, which has strict time limit and any task miss its deadline may cause catastrophic consequence. The existing results for exact schedulability analysis for fixed priority scheduling needs to calculate the response time of each task to check if there is a task with response time greater than its deadline or period.

The most common used fixed priority scheduling is Rate Monotonic (RM) algorithm which was first proposed by Liu and Layland [1]. For RM scheduling, the task with lower period is assigned higher priority. However, the RM algorithm is only applicable to a simple restricted form of fixed priority preemptive scheduling. In order to satisfy the complex requirements of real time systems, much subsequent work has built upon the model presented by Liu and Layland [1] and have successively weakened its constraints and thus widening its applicability. Another common used fixed priority scheduling is Deadline Monotonic (DM) [2] algorithm, which can make the task assign the highest priority with the shortest deadline. .

In scheduling theory, a real time system comprises a set of real time tasks; each task consists of an infinite or finite stream of jobs. In the fixed priority scheduling system, the task set can be scheduled by both RM algorithm and DM algorithm. The success of a real-time system depends on whether all the jobs of all the tasks can be guaranteed to complete their executions before their deadlines. If they can, it can be said that the task set is schedulable.

Schedulability tests can be sufficient or exact (necessary and sufficient). For fixed priority scheduling system, many utilization based tests have been proposed. However, traditional utilization based tests has two obvious drawbacks. One is they are sufficient (that is not exact), another is they are not really apply to the general system model. While the exact schedulability analysis makes up for these drawbacks, which needs to calculate the worst-case response time of each task to judge whether a system is schedulable, which means peiple must analyze each task respectively.

In this paper, firstly, a review of exact schedulability analysis is given for fixed priority preemptive scheduling on uniprocessor. Then the paper provides a new conclusion on context switching overhead and finally this overhead is applied to exact schedulability analysis. The effect of release jitter on task response time is also considered.

In this paper, the task τ_i in each task set is characterized by period T_i, the worst case execution time C_i and relative deadline D_i. We let R_i to be the worst-case response time of task τ_i, which is the longest time of task τ_i from the time it arrives until the time it completes its required computation. At any time, an arrived job with a higher priority can preempt a lower priority job's execution. When a job completes its execution, the system chooses the pending job with the highest priority to execute.

Some real-time systems have tasks that behave as sporadically periodic tasks [3], that is a task arrives at some time, executes periodically for a bounded number of periods (called inner periods), and then does not re-arrive for a longer time (called the outer period). Such as interrupt handlers for bursty interrupts (for example, packet arrivals from a communications device), or certain monitoring tasks [4]. In this paper, the existing results on

exact schedulability analysis are also described for fixed priority system in which tasks are sporadically periodic.

The rest of this paper is organized as follows. Section II describes the existing results on exact schedulability analysis for fixed priority systems with arbitrary deadlines on uniprocessor. Both preemptive scheduling with tasks which are periodic or sporadic periodic and preemptive-threshold scheduling are described. In Section III, a review of scheduling overheads is given and the improvement is provided.. In section IV, the new equations are put forward to calculate the worst-case response time by combing context switching time and release jitter time. Summary is provided in Section V.

II. Previous Results on Exact Schedulability Analysis

This section describes the previous research results on exact schedulability analysis for fixed priority scheduling on uniprocessor. The exact schedulability analysis is based on the calculation of task worst-case response time. These results are described from two aspects, the fixed priority preemption scheduling and the fixed priority preemption-threshold scheduling. For fixed priority preemptive scheduling, two system models are also considered, which are the tasks which are periodic and sporadic periodic.

A. Fixed Priority Preemptive Scheduling

1) Periodic Tasks

In 1986, Joseph and Pandya [5] first proposed the exact schedulability analysis based on task response time. In their analysis, the critical instant was assumed where all tasks are assumed to be released together, which is also the worst-case scheduling scenario for simple tasks. Lately, exact schedulability analysis for rate monotonic algorithm have been independently derived [6][3]. Using the response time analysis proposed by Audsley et al. [3], the periodic task set is schedulable with the RM algorithm if and only if the worst-case response time of each task is less than or equal to its deadline. The worst-case response time R_i of task τ_i can be derived using the following iterative formula:

$$\begin{cases} R_i^{(0)} = C_i \\ R_i^{(n+1)} = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^{(n)}}{T_j} \right\rceil C_j \end{cases} \quad (1)$$

where $hp(i)$ denotes the set of tasks with higher priorities than task τ_i. The worst-case response time of task τ_i is given by the smallest value of $R_i^{(n)}$ such that $R_i^{(n)} = R_i^{(n-1)}$. For the constrained task set (that is $D_i \le T_i$), if all tasks meets their deadlines (that is $R_i^{(n)} \le D_i \ \ i = 1, \cdots, n$), then the task set is schedulable.

Sjödin and Hansson [7] provided some methods for reducing the number of iterations in computing the tasks response time.

In 1990, Lehoczky et al. [8] described qualitative analysis which can determine the worst-case response time of a given task with arbitrary deadlines. They also pointed out that neither the rate monotonic nor deadline monotonic priority ordering policies are optimal for tasks with arbitrary deadlines.

In 1992, Tindell et al. [4] extended the above response time analysis, provided an exact schedulability test for tasks with arbitrary deadlines. The worst-case response time of task τ_i was derived by the following theorem.

Theorem 1 ([4]) The worst-case response time of task τ_i is given by:

$$R_{i,q} = \max\left(w_i(q) - qT_i\right) \quad q = 0,1,2,\cdots, \quad (2)$$

where $w_i(q)$ is given by:

$$w_i(q) = (q+1)C_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i(q)}{T_j} \right\rceil C_j,$$

where B_i in Theorem 1 is the blocking factor of task τ_i, which is the longest time a higher priority task can be delayed by the execution of lower priority tasks. If any special explanations are given, the symbol B_i will be applied to the following analysis.

Burns et al. [9] extended the model in [4] by introducing the two phases of computation and two response times while keep the task priority during execution. The two phases are merely for the purpose of analysis.

2) Sporadically Periodic Tasks

In 1994, Tindell et al. [4] proposed the exact schedulability analysis for fixed priority preemptive scheduling with tasks behave as sporadically periodic. The exact schedulability analysis is also based on the calculation of task worst-case response time. The difference from the previous results is that the tasks here are not periodic but sporadically periodic. In order to give out the calculation of task worst-case response time, they also placed some restrictions on the model that the sporadically periodic task must be finished before its next arrival (i.e. $n_i t_i \le T_i'$), and t_i is the inner period of task τ_i, T_i' is the outer period of task τ_i. The worst-case response time is given as Theorem 2.

Theorem 2 ([4]) The worst-case response time of task τ_i is given by

$$R_i = \max_{q=0,1,2,\cdots} (w_i(q) - m_i t_i - M_i T_i') \quad (3)$$

where m_i is given by:

$m_i = q - n_i M_i$,

and

$M_i = \left\lfloor \frac{q}{n_i} \right\rfloor$,

and where $w_i(q)$ is given by:

$$w_i(q) = (n_i M_i + m_i + 1)C_i + B_i + \sum_{\forall j \in hp(i)} \left(\min\left(n_j, \left\lceil \frac{w_i(q) - F_j T_j'}{t_j} \right\rceil \right) + n_j F_j \right) C_j,$$

and F_j is given by:

$$F_j = \left\lceil \frac{w_i(q)}{T_j} \right\rceil - 1$$

Tindell et al. [4] also considered the release jitter that may suffered by the tasks, and assumed that the release jitter J_i of task τ_i is the inner jitter (i.e. each innovation of task τ_i can suffer release jitter).

B. Fixed Priority Preemption-Threshold Scheduling

The preemptive scheduling, non-preemptive scheduling and preemption-threshold scheduling are the current available fixed priority scheduling algorithms. The preemption-threshold scheduling algorithm is proposed in recent years which has combined the advantage of both preemptive scheduling and non-preemptive scheduling. In this part, a brief description of preemption-threshold scheduling is given and then the improvement is given in Section IV.

Fixed priority preemption-threshold scheduling was first introduced in a commercial real-time operating system ThreadX [10] and the worst-case response time analysis for preemption-threshold scheduling was first presented in [11]. The analysis in [11] was proved optimal in [12]. In 2010, Keskin [13] showed that the existing worst-case response time analysis for preemption-threshold scheduling is pessimistic and presented an exact worst-case response time analysis. For the basic task model described in [13], the worst-case response time for task τ_i is given by Theorem 3.

Theorem 3 ([13]) The worst-case response time R_i of task τ_i is given by

$$R_i = \max(F_{ik} + J_i - kT_i), (\forall k : 0 \le k < l_i) \quad (4)$$

Where J_i is the release jitter and l_i is the maximum number of jobs of task τ_i in a $level-i$ active period which is given by

$$l_i = \left\lceil \frac{L_i}{T_i} \right\rceil,$$

and L_i is the worst-case length of a $level-i$ active period which is given by

$$L_i = B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{L_i + J_j}{T_j} \right\rceil C_j,$$

and F_{ik} is the worst-case finish time of the k^{th} job of task τ_i which is given by

$$F_{ik} = S_{ik} + C_i + \sum_{\forall j \in hp(i)} \left(\left\lceil \frac{F_{ik} + J_j}{T_j} \right\rceil - \left(1 + \left\lfloor \frac{S_{ik} + J_j}{T_j} \right\rfloor \right) \right) C_j,$$

and S_{ik} is the worst-case start time of the k^{th} job of task τ_i that is given by

$$S_{ik} = \begin{cases} B_i + kC_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{S_{ik} + J_j}{T_j} \right\rceil C_j & if\ B_i > 0 \\ kC_i + \sum_{\forall j \in hp(i)} \left(1 + \left\lfloor \frac{S_{ik} + J_j}{T_j} \right\rfloor \right) C_j & if\ B_i = 0 \end{cases}$$

However, the exact response-time analysis for fixed priority preemption-threshold scheduling described above have not considered the task context switching overhead, this is another issue which will be the supplement in Section IV.

III. SCHEDULING OVERHEADS

This section gives a review of some existing results on scheduling overhead firstly, then the improvement will be provided, which will be applied to exact schedulability analysis for fixed priority scheduling presented in s\Section Ⅱ.

In 1995, Audsley et al. [14] considered the effect of context switching overhead on system scheduling. The context switching overhead represents the overhead associated with preempting current task, saving its context, loading the context of the next task and resuming the next task. The most common way of accounting for context switching overhead is to increase the execution time of each task. The execution time for task τ_i becomes $C_i^{'} = C_i + C_{in} + C_{out}$ [15]. Where C_{in} and C_{out} are the worst-case context switching times in and out of any task.

Klein et al. [16] provided a new execution time by assuming that in and out overheads are equal, and the new execution time becomes $C_i^{'} = C_i + 2C_{cs}$.

Craciunas et al. [17] described the overhead by considering two aspects, one is to allow an action to execute for less time than its actual limit within one period and use the remaining time to account for the scheduler overhead. The other aspect is to increase the limit such that the action will be able to execute its original limit and the time spent on scheduler invocations within one period. Simply means that the first method increases the response-time bounds, and the second increases the utilization of an action. By considering a trade-off between an increases in response time versus utilization, they gives the overhead as follows:

$$\delta_{i,j} = \delta_{i,j}^b + \delta_{i,j}^u,$$

where $\delta_{i,j}^b$ is the overhead that extends the response-time bounds of the respective action and $\delta_{i,j}^u$ increases the utilization. From which, the total scheduler overhead for one period of action $\alpha_{i,j}$ is

$$\delta_{i,j} = N_{i,j} \square \xi.$$

Hence, the total overhead is made up of $N_{i,j}$ pieces of ξ workload, which ξ denotes the duration of a single invocation. However, they had limited the task set nonpreemptive.

However, as it is known, the context switching requires to save the task context and loading the task context required for the next task. In order to more accurately

reflect the context switching time of each task, we denote C_i^{sched}, C_i^{save}, C_i^{load} as the time to execute the scheduling code to determine the next task to run, the time to save the state of current task and the time to load the next task respectively. The final task execution time is a simple time cumulation of each part that constitute the context switching. The final task execution time is denoted as the following equation.

$$C_i^{new} = C_i + C_i^{sched} + C_i^{save} + C_i^{load} \quad (5)$$

In the following discussion, we will show the effect of context switching overhead on exact schedulability analysis of fixed priority scheduling. The provided context switching time will be increased to the task execution time.

IV. Exact Schedulability Analysis with Scheduling Overheads

This section shows the effect of scheduling overheads on exact schedulability analysis. The existing results on traditional schedulability analysis always assumed that as soon as a task arrived it is released. But this is not always the case (a task may be delayed because of awaiting the arrival of a message). The difference between the arrival of a task and its release was called release jitter [3][4]. In the following discussion, the release jitter of task τ_i will be denoted by J_i. For the task with deadline which is less than or equal to period, the paper extends the conclusion of the worst-case response time calculation in [3][5][6] by giving the following theorem.

Theorem 4 The worst-case response time R_i of task τ_i is given as the following recursion formula:

$$\begin{cases} R_i^{(0)} = C_i + C_i^{new} \\ R_i^{(n+1)} = C_i + C_i^{new} + \sum_{\forall j \in hp(i)} \left\lceil \dfrac{R_i^{(n)} + J_j}{T_j} \right\rceil \left(C_j + C_j^{new} \right) \end{cases} \quad (6)$$

where $C_i^{new} = C_i + C_i^{sched} + C_i^{save} + C_i^{load}$. As the formula denotes, the fix-point iteration starts with $R_i^{(0)} = C_i$ and terminates when $R_i^{(n)} = R_i^{(n+1)}$ at which the worst-case response time R_i of task τ_i is found. In general, there may be several values of R_i that fit, but the smallest value is the one we want. The value of R_i is valid only if $R_i \le T_i$. This is because if the response time goes over the end of the period then task τ_i could be delayed by a previous invocation of itself. This means the deadline D_i of task τ_i should not be greater than period T_i (i.e. we must have $D_i \le T_i$).

The exact schedulability analysis can be extended to arbitrary deadlines. When deadline is less than or equal to period, it is only necessary to consider a single release of each task. For arbitrary deadline scheduling, a number of releases must be considered. The worst-case response time proposed in [4] can be rewritten by the following equation:

$$w_i(q) = (q+1)\left(C_i + C_i^{new}\right) + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i(q) + J_j}{T_j} \right\rceil \left(C_j + C_i^{new}\right) \quad (7)$$

Where $C_i^{new} = C_i + C_i^{sched} + C_i^{save} + C_i^{load}$ and $hp(i)$ is the set of tasks with priority higher than τ_i. For each value of q, the equation above can be solved by forming a recurrence relation on $w_i(q)$. The corresponding response time is then given as

$$R_{i,q} = w_i(q) - qT_i$$

The number of releases that need to be considered is bounded by the lowest value of q for which the following relation is true:

$$R_{i,q} \le T_i$$

At this point the task completes its execution before the next release and the worst-case response time is then given as follows:

$$R_i = \max R_{i,q} \quad q = 0,1,2,\cdots$$

So far the paper has assumed that the task sets consist of periodic tasks. In general this is not always the case. Just as we have described in part 2) of Section II, the interrupt handlers for bursty interrupts (for example, packet arrivals from a communication device), or certain monitoring tasks are such sporadically periodic tasks.

In order to give out the worst-case response time for sporadically periodic tasks, Tindell et al. [4] also proposed a simplified system model for sporadically periodic task systems. In order to propose the improvement conclusion, a brief description of the system model will be given again here.

The task τ_i in sporadically periodic task system is usually assigned two periods: the inner period which is denoted as t_i and the outer period which is denoted as $T_i^{'}$. The inner period is the worst-case inter-arrival time between tasks within a burst while the outer period is the worst-case inter-arrival time between bursts. The number of arrivals to each burst are bounded by a certain value, the total time for the burst (that is, the number of inner arrivals multiplied by the inner period) must be less than or equal to the outer period.

In Theorem 5, the worst-case response time calculation was extended which was provided in [4] by considering scheduling overhead for sporadically periodic.

Theorem 5 The worst-case response time R_i of task τ_i is given by

$$R_i = \max(w_i(q) + J_i - m_i t_i - M_i T_i) \; q = 0,1,2,3,\cdots, \quad (8)$$

where m_i is given by

$$m_i = q - M_i n_i,$$

n_i is the number of inner arrivals with a burst for task τ_i and

$$M_i = \left\lfloor \frac{q}{n_i} \right\rfloor,$$

and where $w_i(q)$ is given by

$$w_i(q) = (M_i n_i + m_i + 1)(C_i + C_i^{new}) + B_i + \sum_{\forall j \in hp(i)} \left(\min\left(n_j, \left\lceil \frac{J_j + w_i(q) - F_j T_j}{t_j} \right\rceil \right) + F_j n_j \right) (C_j + C_j^{new})$$

,

and F_j is given by

$$F_j = \left\lceil \frac{J_j + w_i(q)}{T_j} \right\rceil - 1$$

.

From the above equation, when evaluating the worst-case response time for task τ_i, the iteration can stop as soon as $w_i(q) \le M_i T_i + m_i t_i - J_i$.

The fixed priority preemption-threshold scheduling has been proposed as a generalization of fixed priority preemptive scheduling, which has imporved the schedulability to a great degree. The existing results for exact response-time analysis for fixed priority preemptive-threshold scheduling without considering task context switching time has weaken its accuracy to some extent. This is why a supplement will be made by considering task context switching time that is described in Section III. The extended conclusion will be given directly on the basis of [13] which is also described in part B of Section II. The conclusion is denoted as Theorem 6.

Theorem 6 The worst-case response time R_i of task τ_i is given by

$$R_i = \max(F_{ik} + J_i - kT_i), (\forall k : 0 \le k < l_i) \quad (9)$$

Where l_i is the maximum number of jobs of task τ_i in a $level-i$ active period which is given by

$$l_i = \left\lceil \frac{L_i}{T_i} \right\rceil$$

,

and L_i is the worst-case length of a $level-i$ active period which is given by

$$L_i = B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{L_i + J_j}{T_j} \right\rceil (C_j + C_j^{new})$$

,

and F_{ik} is the worst-case finish time of the k^{th} job of task τ_i which is given by

$$F_{ik} = S_{ik} + (C_i + C_i^{new}) + \sum_{\forall j \in hp(i)} \left(\left\lceil \frac{F_{ik} + J_j}{T_j} \right\rceil - \left(1 + \left\lfloor \frac{S_{ik} + J_j}{T_j} \right\rfloor \right) \right) (C_j + C_j^{new})$$

,

and S_{ik} is the worst-case start time of the k^{th} job of task τ_i that is given by

$$S_{ik} = \begin{cases} B_i + k(C_i + C_i^{new}) + \sum_{\forall j \in hp(i)} \left\lceil \frac{S_{ik} + J_j}{T_j} \right\rceil (C_j + C_j^{new}) & if\ B_i > 0 \\ k(C_i + C_i^{new}) + \sum_{\forall j \in hp(i)} \left(1 + \left\lfloor \frac{S_{ik} + J_j}{T_j} \right\rfloor \right) (C_j + C_j^{new}) & if\ B_i = 0 \end{cases}$$

V. SUMMARY

The schedulability analysis based on worst-case response time is an important work than the common used utilization based test because it provides an exact schedulability analysis which needs to calculate the worst-case response time of each task. Once there is a worst-case response time for each task, the schedulability test is simple: we just look to see that if $R_i \le D_i$ meets for each task. Exact schedulability analysis can be extended to arbitrary deadlines. As a generalization of fixed priority preemptive scheduling, fixed priority preemptive-threshold scheduling can improve the schedulability. The exact schedulability analysis with scheduling overhead and release jitter will more accurately reflected the observed behavior of the system.

REFERENCES

[1] C. L. Liu and J. W. Layland, “Scheduling algorithms for multiprogramming in a hard real-time environment,” Journal of the ACM, vol. 20, no. 1, pp. 40–61, 1973.

[2] J. Y.-T. Leung and J. Whitehead, “On the complexity of fixed-priority scheduling of periodic, real-time tasks,” Performance Evaluation, vol. 2, pp. 237–250, 1982.

[3] N. C. Audsley, A. Burns, K. Tindell, and A. Wellings, “Applying new scheduling theory to static priority preemptive scheduling,” Software Engineering Journal, December 1993.

[4] K. Tindell, A. Burns and A.J. Wellings, “An Extendible Approach for Analysing Fixed Priority Hard Real-Time Tasks”, Real-Time Systems 6(2), pp. 133-151 (March 1994).

[5] M. Joseph and P. Pandya, “Finding response times in a real-time system,”The Computer Journal, British Computer Society, vol. 29, no. 5, pp.390–395, October 1986.

[6] J. P. Lehoczky, L. Sha, and Y. Ding, “The rate-monotonic scheduling algorithm: Exact Characterization and average case behavior,” in Proceedings of the 10th IEEE Real-Time Systems Symposium, 1989, pp.166–172.

[7] M. Sj¨odin and H. Hansson, “Improved response-time analysis calculations,” in Proceedings of the 19th IEEE Real-Time Systems Symposium, December 1998, pp. 399–409.

[8] J. P. Lehoczky, "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines," in RTSS, 1990, pp. 201-209.

[9] A. Burns, K. Tindell, and A. Wellings. "Fixed priority scheduling with deadlines prior to completion." Real-Time Systems, 1994. Proceedings., Sixth Euromicro Workshop on. IEEE, 1994.

[10] Express Logic Inc. ThreadX Technical Features, v5.1. [Online].Available: http://rtos.com/products/threadx, June 2010.

[11] Y. Wang and M. Saksena, "Scheduling fixed-priority tasks with preemption threshold," in Real-Time Computing Systems and Applications, 1999. RTCSA'99. Sixth International Conference on, 1999, pp. 328-335.

[12] J. Regehr, "Scheduling tasks with mixed preemption relations for robustness to timing faults," in Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE, 2002, pp. 315-326.

[13] U. Keskin, R. J. Bril, and J. J. Lukkien, "Exact response-time analysis for fixed-priority preemption-threshold scheduling," in

Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on, 2010, pp. 1-4.

[14] N. C. Audsley, A. Burns, R. I. Davis, K. W. Tindell, A. J. Wellings. Fixed priority pre-emptive scheduling: an historical perspective. Real-Time Syst., vol. 8(2-3):pp. 173−198, 1995.

[15] D. I. Katcher, H. Arakawa, J. K. Strosnider. Engineering and analysis of fixed priority schedulers. IEEE Trans. Softw. Eng., vol. 19(9):pp. 920-934, 1993.

[16] M. H. Klein, T. Ralya. An analysis of input/output paradigms for real-time systems. Tech. Rep. CMU/SEI-90-TR-19, Carnegie-Mellon University, 1990.

[17] S. S. Craciunas, C. M. Kirsch, and A. Sokolova, "Response time versus utilization in scheduler overhead accounting," in Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE, 2010, pp. 291-300.

[18] J. Mäki-Turja, M. Nolin: Faster Response Time Analysis of Tasks With Offsets. In: Proc. 10th IEEE Real-Time Technology and Applications Symposium (RTAS). (2004), 2006.

[19] K. Tindell, Addind Time-Offsets to Schedulability Analysis, Technical Report YCS 221, Dept of Computer Science, University of York, England, January 1994.