Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information Processing Letters 97 (2006) 83–87
Information Processing Letters
www.elsevier.com/locate/ipl

Runtime feasibility check for non-preemptive real-time periodic tasks ☆

Sangwon Kim *, Joonwon Lee, Jinsoo Kim

Division of Computer Science, Department of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology, 373-1 Guseong-dong, Yuseong-gu, Daejeon 305-701, Republic of Korea

Received 18 July 2005; received in revised form 5 October 2005; accepted 13 October 2005

Available online 28 November 2005

Communicated by A.A. Bertossi

Keywords: Real-time systems; Non-preemptive scheduling

1. Introduction

The most important requirement for real-time systems is the capability to support the timely execution of applications. Real-time scheduling over a single processor has been widely studied in the last thirty years, and it has been proven that the non-preemptive version of Earliest Deadline First (EDF) algorithm is optimal [1], in the sense that if a set of tasks is schedulable, it is also schedulable using EDF. EDF scheduler always selects and executes a task having the earliest deadline. Most non-preemptive real-time systems favor the EDF scheduler for its easy feasibility check and optimal schedulability.

In EDF scheduling, however, scheduler leaves no chance for other feasible alternative task sequences to be selected. Fig. 1 shows an example in which an alternative task execution sequence is more preferable than that of EDF scheduling. In the example, device power mode can transit from 'active' to 'sleep' and vice versa in order to save power during idle time. A better task sequence can be generated by eliminating unnecessary power-mode transitions and increasing the idle time of devices. Such a task sequence can be easily obtained in a non real-time system by scheduling first those tasks that use devices in active mode [6]. However, in a real-time system, the timely execution of tasks must be guaranteed before scheduling a task in a non-EDF manner (i.e., scheduling a task not having earliest deadline first). Therefore, in [7], an off-line scheduling algorithm is presented to generate the optimal task sequence that minimizes energy consumption of I/O devices without deadline misses. Nevertheless, finding the optimal task sequence is impractical since the hyper-period of a general task set is too long to be optimized.

Another example in which task sequence makes a difference is presented in [3]. Here, the execution time of tasks can be reduced by scheduling continuously the tasks that share common data set and increasing the cache hit ratio. In [2], it is shown that the saved time (dynamic slack) can be used to minimize the energy consumption of a processor in a non-preemptive real-time system.

☆ This research was supported by the MIC (Ministry of Information and Communication), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Assessment) (IITA-2005-C1090-0502-0031).

* Corresponding author.
E-mail address: kimsw@calab.kaist.ac.kr (S. Kim).

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2005.10.010

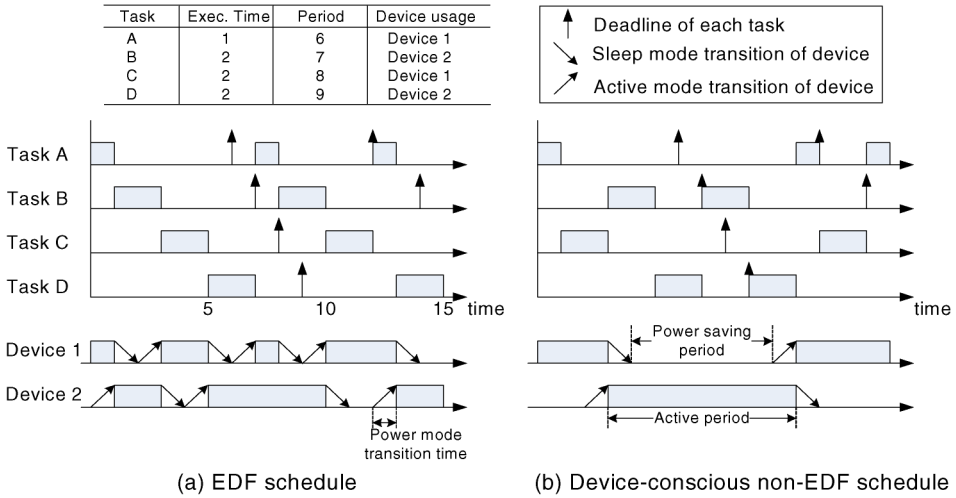

Fig. 1. Non-EDF scheduling for device power saving.

Therefore, based on the given system purpose in addition to the timely execution of tasks, there can be a preferred task for scheduler to generate a better task sequence. To support such task preferences at runtime and to prevent future deadline misses, the feasibility of future tasks must be guaranteed before scheduling a task in a non-EDF manner.

This paper presents an online algorithm that checks the feasibility of a task set in order to schedule a task in a non-EDF manner. This algorithm can be used by a real-time system to schedule periodic tasks over a single processor.

The rest of this paper is organized as follows. Section 2 describes the system model and notations briefly. In Section 3, we present the runtime feasibility checking algorithm and prove that a task scheduled by the algorithm does not make a deadline miss. A preliminary experiment results follow in Section 4 and we conclude in Section 5.

2. System model and notations

The scheduling of a task set is said to be **valid** if and only if no task instance misses its deadline. And a task set is said to be **feasible** with respect to a given class of schedulers if and only if there is at least one valid schedule that can be obtained by a scheduler of this class.

In this paper, we restrict ourselves to the case in which the system schedules tasks non-preemptively on a single processor, and consists of a feasible task set of n periodic real-time tasks, denoted as $\tau = \{T_1, T_2, \ldots, T_n\}$. A task T_i is a pair (c_i, p_i) where

- c_i: The computational cost; The amount of processor time required to execute T_i, and
- p_i: The period; The minimal interval between invocations of T_i.

We assume that a relative deadline of each task is equal to the period of the task. D_i is the absolute deadline of the latest invocation of T_i. If T_i is invoked at time t, then D_i is $t + p_i$. $s_i(t)$ is the status of the last invocation of T_i at time t. $s_i(t)$ can be set to one of three states: 'Invoked', 'Running' and 'Done'.

3. Runtime feasibility check

In this section, we define laxity of a task and use it to check the feasibility of a task set when scheduling a task in a non-EDF manner. For a feasible task set, the feasibility of the task set after scheduling T_i in a non-EDF manner can be guaranteed by checking a deadline miss within D_i, as stated (without proof) in the following Theorem 1.

Theorem 1. *For a feasible periodic task set $\tau = \{T_1, T_2, \ldots, T_n\}$, scheduling a task T_i in a non-EDF manner at some scheduling point in time can make valid schedule of τ if and only if it does not make any deadline miss before D_i.*

In the following, we define the laxity of each task and then explain how to use it for runtime feasibility check.

Definition 1. For a periodic task set $\tau = \{T_1, T_2, \ldots, T_n\}$, where $T_i = (c_i, p_i)$, sorted in nondecreasing order

```
1: On invocation of T_i:
2: L_i = Laxity_i − (t_current − t_{i_invoke});
3: for j = 1 to n do
4:    if s_j(t_{i_invoke}) = 'Invoked' and D_j ⩽ D_i then
5:       L_i = L_i − c_j;
6:    end if
7: end for
8: On completion of T_i:
9: for j = 1 to n do
10:   if s_j(t_current − c_i) = 'Invoked' and D_j < D_i then
11:      L_j = L_j − c_i;
12:   end if
13: end for
```

Algorithm 1. Runtime laxity adjustment.

of the task period, $Laxity_i$ of T_i denotes the maximum value of **L** satisfying the following condition:

$$\forall t,\ p_1 \leqslant t \leqslant p_i: \quad t \geqslant c_i + \sum_{j=1,\ j\neq i}^{n} \left\lfloor \frac{t}{p_j} \right\rfloor \cdot c_j + \mathbf{L}. \tag{1}$$

$Laxity_i$ can be computed off-line. Considering only tasks that will be invoked and completed during p_i by EDF scheduling, $Laxity_i$ is surplus time that can be used by any other tasks without the deadline miss of a task that has an invocation and a deadline during p_i.

However, there can be tasks that lead $Laxity_i$ to be adjusted at runtime. Tasks can be classified into three different classes with respect to T_i; tasks invoked prior to the invocation of T_i and have deadline before D_i, tasks completed after the invocation of T_i and have deadline after D_i, and other remaining tasks. Two runtime adjustments of $Laxity_i$ is required regarding the execution of the first two classes of tasks as shown in Algorithm 1. The first runtime adjustment of $Laxity_i$ is done at the first scheduling time after invocation of T_i as shown in lines 1–7 of Algorithm 1. Note that t_{current} and $t_{i_{\text{invoke}}}$ denote the point in time at which Algorithm 1 is executed and T_i is invoked, respectively. The second runtime adjustment of $Laxity_i$ is done after completion of a task as shown in lines 8–13. All the tasks that yielded their execution chance to the completed one must have their runtime laxity value readjusted.

Now, before scheduling T_i in a non-EDF manner, the feasibility of the task set after scheduling T_i must be checked using the routine shown in Algorithm 2.

Theorem 2. *For a feasible task set $\tau = \{T_1, T_2, \ldots, T_n\}$, where $T_i = (c_i, p_i)$, the task set τ after scheduling a task in a non-EDF manner using Algorithms* 1 *and* 2 *is also feasible.*

```
1: On checking feasibility
      (to check validity of scheduling T_i):
2: if L_i < 0 then
3:    return(fail) {Cannot guarantee feasibility};
4: end if
5: for j = 1 to n do
6:    if s_j(t_current) = 'invoked' and D_j < D_i and
         i ≠ j and L_j < c_i then
7:       return(fail) {Cannot guarantee feasibility};
8:    end if
9: end for
10: return(success) {Can guarantee feasibility};
```

Algorithm 2. Runtime feasibility check.

Proof. In [1], it is proved that EDF-Scheduling is valid for a feasible task set. Therefore we only need to prove that the task scheduled in a non-EDF manner that passed the feasibility check presented by Algorithm 2 does not make any deadline misses for the feasible task set.

Let us define and use the following notations for simple proof.

- t_{miss}: the earliest point in time at which a deadline is missed.
- T_m: the task that made the deadline miss at t_{miss}.
- T_i: the last task that is scheduled before t_{miss} with deadline after t_{miss}, in a non-EDF manner using Algorithms 1 and 2.
- t_{sched}: the point in time at which T_i was scheduled immediately prior to t_{miss}.
- $t_{k_{\text{invoke}}}$: the point in time at which a task T_k was invoked immediately prior to t_{miss}.
- $t_{k\text{-sched}}$: the first scheduling point in time after $t_{k_{\text{invoke}}}$.

And let us define the following sets of tasks from τ.

- G_1^k = the set of tasks invoked before $t_{k_{\text{invoke}}}$ with a deadline before t_{miss} and executed after $t_{k_{\text{invoke}}}$,
- G_2^k = the set of tasks invoked before $t_{k_{\text{invoke}}}$ with a deadline after t_{miss} and executed in $[t_{k_{\text{invoke}}}, t_{\text{miss}}]$,
- G_3^k = the set of tasks invoked after $t_{k_{\text{invoke}}}$ with a deadline after t_{miss} and executed before t_{miss}.

By Theorem 1, the tasks scheduled in a non-EDF manner after the execution of T_i do not make the deadline miss at t_{miss} because they must have their deadlines before t_{miss}. Therefore, there are two cases to consider from the relation between $T_{m_{\text{invoke}}}$ and $T_{i_{\text{invoke}}}$.

• *Case* 1: T_m was invoked before t_{sched} ($t_{m_{\text{invoke}}} < t_{\text{sched}}$).

The processor demand in $[t_{m_{\text{invoke}}}, t_{\text{miss}}]$, $d_{t_{m_{\text{invoke}}}, t_{\text{miss}}}$, is bounded by

$$d_{t_{m_{\text{invoke}}},t_{\text{miss}}} \leqslant c_m + (t_{m_{\text{sched}}} - t_{m_{\text{invoke}}}) + \sum_{j=1,\, j\neq m}^{n} \left\lfloor \frac{t_{\text{miss}} - t_{m_{\text{invoke}}}}{p_j} \right\rfloor \cdot c_j + \sum_{T_k \in G_1^m} c_k + \sum_{T_l \in G_2^m} c_l + \sum_{T_u \in G_3^m} c_u. \quad (2)$$

L_m, the runtime laxity value of T_m, is adjusted for the execution time of the task that was in execution at $t_{m_{\text{invoke}}}$ by line 2 of Algorithm 1. It is readjusted for the execution time of the tasks in G_1^m by lines 3–7. After execution of the tasks in G_2^m or G_3^m, it is adjusted again by lines 9–13 of Algorithm 1.

And since T_i is the last task that is scheduled before t_{miss} with deadline after t_{miss}, the last adjustment of L_m was done at t_{sched}. As T_i passed the feasibility check in Algorithm 2 with c_i less than L_m, $L_m \geqslant 0$, at t_{miss}. Hence, at t_{miss},

$$L_m = \mathit{Laxity}_m - (t_{m_{\text{sched}}} - t_{m_{\text{invoke}}}) - \sum_{T_k \in G_1^m} c_k - \sum_{T_l \in G_2^m} c_l - \sum_{T_u \in G_3^m} c_u \geqslant 0. \quad (3)$$

From (2) and (3), and since we have a deadline miss at t_{miss}, we can write the following inequality

$$t_{\text{miss}} - t_{m_{\text{invoke}}} < d_{t_{m_{\text{invoke}}},t_{\text{miss}}} \leqslant c_m + \sum_{j=1,\, j\neq m}^{n} \left\lfloor \frac{t_{\text{miss}} - t_{m_{\text{invoke}}}}{p_j} \right\rfloor \cdot c_j + \mathit{Laxity}_m.$$

This is a contradiction to (1) and the feasibility condition of a non-preemptive task set [1].

• *Case* 2: T_m was invoked after t_{sched} ($t_{\text{sched}} \leqslant t_{m_{\text{invoke}}}$).

There could be no idle time in $[t_{i_{\text{invoke}}}, t_{\text{sched}}]$ because the presented scheduling algorithm is work conserving. Therefore, the processor demand in $[t_{i_{\text{invoke}}}, t_{\text{miss}}]$ making deadline miss at t_{miss} can be presented as below

$$t_{\text{miss}} - t_{i_{\text{invoke}}} < d_{t_{i_{\text{invoke}}},t_{\text{miss}}} \leqslant c_i + (t_{i_{\text{sched}}} - t_{i_{\text{invoke}}}) + \sum_{j=1,\, j\neq i}^{n} \left\lfloor \frac{t_{\text{miss}} - t_{i_{\text{invoke}}}}{p_j} \right\rfloor \cdot c_j + \sum_{T_k \in G_1^i} c_k + \sum_{T_l \in G_2^i} c_l + \sum_{T_u \in G_3^i} c_u. \quad (4)$$

L_i, the runtime laxity value of T_i, cannot be less than 0 at t_{miss}, because T_i passed line 2 of Algorithm 2 and T_i was the last task scheduled before t_{miss} with deadline after t_{miss}. Since (4) has the same form as (2), this is also a contradiction to (1) and the feasibility condition of a non-preemptive task set [1]. □

4. Preliminary experiments

We conducted some preliminary experiments to figure out the level at which a scheduler can support task preference at runtime using the proposed algorithm. We evaluated the algorithm on two real-life task sets, a computer numerical control (CNC) task set [4] and a generic aviation platform (GAP) task set [5], assuming the relative deadline of each task is identical to its period. To generate a runtime task preference, we made some task groups (of 2, 3, 5 and 10) out of task set so that each task belongs to only one group. And we assumed that the system prefers to schedule a task that is in the same group with previously completed one. We counted the number of change of task group in scheduled task sequence and used the counter value as a performance metric of supporting a runtime task preference.

Fig. 2 shows the number of task group changes, normalized to that of EDF scheduling, when CNC and GAP task sets are scheduled using the proposed algorithm. The level of supporting task preference in scheduling increases with small numbers of task groups. This is because the possibility for a preferred task to exist in invoked task list, at scheduling time, increases with a

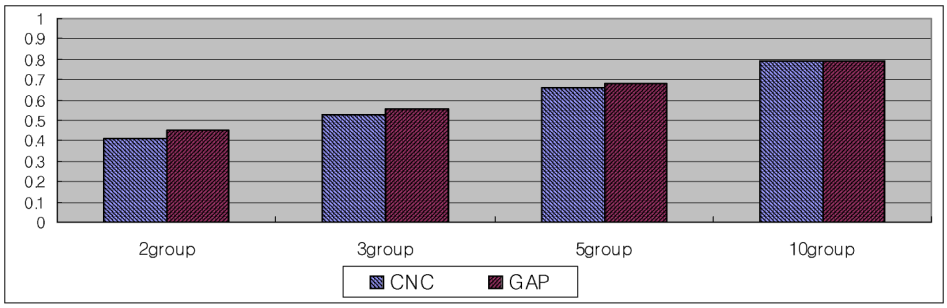

Fig. 2. Normalized number of task group changes at different number of task groups.

small number of task groups. Though the supporting level differs with system environments, it can be easily seen that the proposed algorithm can be useful for real-time schedulers to support a runtime task preference.

5. Conclusion

In this paper we have proposed an algorithm for a scheduler to check, at runtime, the feasibility of a task set before scheduling a task in a non-EDF manner. And we have proved that a non-EDF scheduling using the proposed algorithm guarantees the timely execution of a feasible task set. The proposed algorithm can be useful for real-time systems to support a runtime task preference. It is also valid for systems running a sporadic task set and directly applicable to recent works for energy saving in non-preemptive real-time systems.

References

[1] K. Jeffay, D.F. Stanat, C.U. Martel, On non-preemptive scheduling of periodic and sporadic tasks, in: Proc. of the Real-Time Systems Symposium, 1991, pp. 129–139.
[2] R. Jejurikar, R. Gupta, Energy aware non-preemptive scheduling for hard real-time systems, in: Proc. of 17th of Euromicro Conference on Real-Time Systems, 2005, pp. 21–30.
[3] I. Kadayif, M. Kandemir, I. Kolcu, G. Chen, Locality conscious process scheduling in embedded systems, in: Proc. 10th Internat. Symp. on Hardware/Software Codesign, 2002, pp. 193–198.
[4] N. Kim, M. Ryu, S. Hong, M. Saksena, C. Choi, H. Shin, Visual assessment of a real-time system design: case study on a CNC controller, in: Proc. of the Real-Time Systems Symposium, 1996, pp. 300–310.
[5] D.C. Locke, D. Vogel, T. Mesler, Building a predictable avionics platform in Ada: a case study, in: Proc. of the Real-Time Systems Symposium, 1991, pp. 181–189.
[6] Y.-H. Lu, L. Benini, G. De Micheli, Low-power task scheduling for multiple devices, in: Proc. of International Workshop on Hardware/Software Codesign, 2000, pp. 39–43.
[7] V. Swaminathan, K. Chakrabarty, Pruning-based, energy-optimal, deterministic I/O device scheduling for hard real-time systems, ACM Trans. Embedded Comput. Syst. 4 (1) (2005) 141–167.