Journal of Computing and Information Technology - CIT 18, 2010, 4, 393–400
doi:10.2498/cit.1001879

Imprecise Computation Model, Synchronous Periodic Real-time Task Sets and Total Weighted Error

Damir Poleš[1] and Leo Budin[2]

[1] Eurocontrol Experimental Centre, Brétigny sur Orge, France
[2] Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia

This paper proposes two scheduling approaches, one-level and two-level scheduling, for synchronous periodic real-time task sets based on the Imprecise Computation Model. The imperative of real-time systems is a reaction on an event within a limited amount of time. Sometimes the available time and resources are not enough for the computations to complete within the deadlines, but still enough to produce approximate results. The Imprecise Computation Model is motivated by this idea, which gives the flexibility to trade off precision for timeliness. In this model a task is logically decomposed into a mandatory and optional subtask. Only the mandatory subtask is required to complete by its deadline, while the optional subtask may be left unfinished. Usually, different scheduling policies are used for the scheduling of mandatory and optional subtasks. For both proposed approaches the earliest deadline first and rate monotonic scheduling algorithms are used for the scheduling of mandatory subtasks, whereas the optional subtasks are scheduled in a way that the total weighted error is minimized. The basic idea of one-level scheduling is to extend the mandatory execution times, while in two-level scheduling the mandatory and optional subtasks are separately scheduled. The single preemptive processor model is assumed.

Keywords: Imprecise Computation Model, real-time systems, one-level scheduling, two-level scheduling, total weighted error

1. Introduction

The correctness of many systems and devices in our society depends not only on the logical correctness of results they produce, but also on the time at which they are produced. These real-time systems play an important role and they have been included in the wide range of applications from nuclear power plants, railway systems, air traffic control, military and health monitoring systems to telecommunication networks, multimedia systems, monitoring systems, virtual realty and computer games. Since computers are more often used in our everyday activities, real-time systems will be more important. Scheduling is concerned with the allocation of scarce resources and it is a central activity of a real-time system with the objective to meet specified time constraints. For some real-time systems, all specified timing requirements must be met (hard real-time systems) and a failure to do so may lead to catastrophic consequences, whereas for others some specified timing requirements may be missed (soft real-time systems).

This paper presents the result of an analysis of the multi-level scheduling approach for dynamic real time systems with flexible timing requirements. In particular, it proposes one and two-level scheduling approaches for the Imprecise Computation Model. The imperative is to establish the theoretical lower bound for the total weighted error for both approaches as the most complex error case. This directly leads to the lower bound for the total error and the upper bound for the processor utilization.

Usually, real-time systems are modeled as sets of real-time tasks, $\tau = \{T_1, T_2, \ldots, T_n\}$, which are characterized by a set of parameters including timing requirements that must be met. Scheduling algorithms are used in real-time systems to ensure that each task meets all constraints. A scheduling algorithm is used to create a schedule, S, according to which the tasks will be executed at each time instant by a real-time operating system. A feasible schedule is a schedule in which all tasks meet timing and all other specified constrains.

Sometimes, it is not possible to create a feasible schedule, i.e. it is not possible for all tasks to meet their timing constraints. But, there still may be enough resources to produce approximate results. The generation of partial results using less time and resources is the basis of Imprecise Computation Model. In the Imprecise Computation Model ([3], [4], [6], [8], [9]) each task T_i is logically decomposed into two subtasks: mandatory subtask M_i and optional subtask O_i. Only the mandatory subtask of each task is required to be completed by the task's deadline in order to produce a minimum quality, but still acceptable, result. The optional subtask does not have to be completed (if there is no available processor's time). Similarly, a schedule is defined to be feasible in the Imprecise Computation Model when all mandatory subtasks meet timing and all other specified constraints.

The task set used in the analysis consists of a set of n independent, preemptable tasks, $TS = \{T_1, T_2, \ldots, T_n\}$. Task T_i is characterized by the quintuple $T_i = (r_i, d_i, m_i, o_i, p_i)$ where r_i, d_i, m_i, o_i and p_i denote the task's ready time, relative deadline, mandatory subtask's execution time, optional subtask's execution time and the task's period, respectively. The task's total execution time is defined as $e_i = m_i + o_i$. Appropriate mandatory M and optional O subtask sets consist of n independent, preemtable tasks, $M = \{M_1, M_2, \ldots, M_n\}$ and $O = \{O_1, O_2, \ldots, O_n\}$. Task M_i is characterized by the quadruple $M_i = (r_i, d_i, m_i, p_i)$ and O_i by $O_i = (r_i, d_i, o_i, p_i)$. Each task T_i, M_i and O_i generates an infinite number of jobs T_{ij}, M_{ij}, O_{ij}, $j > 0$, with $r_{ij} = r_i + (j-1)p_i$ and $d_{ij} = d_i + (j-1)p_i$. The task set, TS, is assumed to be synchronous. A task set is said to be synchronous if all ready times are zeros, otherwise a task set is asynchronous. This implies that all ready times in our analysis are assumed to be equal to zero, $r_i = 0$. The tasks' deadlines are assumed to be equal to tasks' periods $d_i = p_i$. Without loss of generality, integer values for all tasks' parameters are assumed [1]. The synchronism of task set and integer parameters facilitate the analysis which, in this case, can be performed on the limited time period which is equal to hyperperiod H, the least common multiplier of all periods in TS, $H = lcm(p_1, \ldots, p_n)$. The single preemptive processor system is assumed and the context switching times are neglected. The processor utilization, $U(TS)$, of task set TS is defined as $U(TS) = \sum_{i=1}^{n} \frac{e_i}{p_i}$.

2. Total Weighted Error

Since the Imprecise Computation Model gives the opportunity to trade-off between the result quality of computations and computation timing requirements, a lot of different metrics have been introduced in order to validate the trade-off and set objectives. Accordingly, a lot of different definitions of errors that correspond to different measures, objectives and applications have been proposed in the literature, e.g. the total weighted error ([6], [4]), the normalized error [8], the maximum normalized error [8], the maximum weighted error [4], the fraction of discarded work [3], number of tardy tasks [6]. It is clear that the error which will be analyzed and objectives that will be optimized depend on an application. Further in this paper, only the total weighted error and its minimization is considered.

In general, the problem of minimizing the total weighted error may be transformed to the minimizing the total weighted tardiness [2] or to a minimum-cost-maximum-flow problem [4]. In the paper, the total weighted error is analyzed for the synchronous periodic task sets with respect to the earliest deadline first (EDF) and rate monotonic (RM) scheduling algorithms used for scheduling of the mandatory subtask sets.

Let S be a feasible schedule for task set TS, and $\alpha_{ij}(T_i, S)$ denote the amount of processing time assigned for the execution of the j$^{\text{th}}$ job of task T_i in schedule S. The amount of discarded work [3], the error, of the j$^{\text{th}}$ job of task T_i is defined to be $\varepsilon(T_{ij}, S) = e_i - \alpha_{ij}(T_i, S)$. For synchronous periodic task set TS and feasible schedule S, the error of task T_i in hyperperiod H is defined as follows:

$$\varepsilon(T_i, S, H) = \sum_{j=1}^{n_i} \varepsilon(T_{ij}, S); \quad n_i = \frac{H}{p_i}; \qquad (1)$$

where n_i is the number of jobs of task T_i occurred in hyperperiod H. Finally, the total weighted error for task set TS and feasible schedule S in hyperperiod H is defined as:

$$\varepsilon = \varepsilon(TS, S, H) = \sum_{i=1}^{n} w_i \varepsilon(T_i, S, H); \qquad (2)$$

where w_i is the weight of error of task T_i. For task set TS it is assumed $w_1 \geq \ldots \geq w_n$. The weight of error determinates the importance of the task. As the weight of error is higher the importance of the task is higher. In case of the total error, all weights are the same.

3. Scheduling

The scheduling of task set TS is based on the usage of the EDF and RM scheduling algorithms for the scheduling of mandatory subtask set. It is assumed that the mandatory subtask set M is schedulable by EDF and RM. Taking into account optional task set parameters, it is obvious that the same scheduling algorithms for the optional subtask set at the same time or level cannot be applied.

Let a scheduling of Imprecise Computation Model in which a scheduler is not able to consider the mandatory subtask set and the optional subtask set separately, i.e. a scheduler applies the same scheduling policy for all tasks in a task set, be called one-level scheduling. On the other side, let a scheduling in which a scheduler is able to consider the mandatory subtask set and the optional subtask set separately, i.e. a scheduler applies separate, possibly different, scheduling policies for mandatory and optional subtask sets, be called two-level scheduling.

In the paper, both approaches are analyzed. In the first approach, the use of one-level scheduling is assumed. The idea is to extend mandatory execution times of all mandatory subtasks in TS as much as possible in a way that the total weighted error of TS is minimized. The result of extension is task set M' which consists of mandatory subtask set only and should be scheduled making use of EDF or RM.

In the second approach, the use of two-level scheduling is assumed and a scheduler must produce schedules on two levels. On the high level the mandatory subtask set is scheduled while on the low level the optional subtask set is scheduled. The high level has to ensure processor time and its assignment to the low level. The low level has to ensure the execution of optional subtask set in a way that the total weighted error of TS is minimized.

3.1. One-level scheduling

One-level scheduling approach is based on the creation of a new mandatory subtask set M' that extends mandatory execution times in comparison to M. The optional sub-task set, O, is not directly scheduled. It is included in the schedule of new mandatory sub-task set, M'. The extension of mandatory subtask set M is done in a way that the total weighted error is minimized. Finally, the schedules S_1 and S_2 of M', generated using EDF and RM, are used for scheduling task set TS. The extending process may be divided in the following steps:

- determination of the maximum extension, ext_{MAX}, of mandatory execution times;
- determination of the extension for each mandatory subtask M_i, ext_i, in a way that the total weighted error is minimized.

The process starts from M and results is the new task set M', $M' = \{M'_1, M'_2, \ldots, M'_n\}$, $M'_i = (r_i, d_i, m'_i, p_i)$, $r_i = 0$, $d_i = p_i$, $m'_i = m_i + ext_i$, $0 \leq ext_i$, $\forall i$.

The maximum amount of time that can be assigned for the extension of mandatory subtasks during hyperperiod H, ext_{MAX}, depends on a scheduling algorithm used. ext_{MAX} must ensure the schedulability of M'.

In the case of the EDF scheduling algorithm the maximum total extension time is defined as:

$$ext_{MAX,EDF} = (1 - U(M))H, \qquad (3)$$

where $U(M)$ is the utilization of mandatory subtask set M. The value of $ext_{MAX,EDF}$ guarantees the utilization of M' to be less or equal to 1, $U(M') \leq 1$, which implies the schedulability of M' [5].

In the case of the RM scheduling algorithm, the maximum total extension time is defined as:

$$ext_{MAX,RM} = (U_{RM,n} - U(M))H, \qquad (4)$$

where $U_{RM,n}$ is the least upper bound to processor utilization for a set of n tasks with fixed priorities, $U_{RM,n} = n(2^{1/n} - 1)$ [5]. Similar as for EDF, the value of $ext_{MAX,RM}$ guarantees the utilization of M' to be less or equal to $U_{RM,n}$, $U(M') \leq n(2^{1/n}-1) = U_{RM,n}$, which implies the schedulability of M' [5]. It is obvious that the sum of all extensions must be less or equal to ext_{MAX}, regardless of the algorithm used:

$$\sum_{i=1}^{n} n_i ext_i \leq ext_{MAX}. \quad (5)$$

The objective to minimize the total weighted error of TS in resulting schedule S' of M' derives the conditions for the determination of mandatory execution time extensions for all tasks, whether EDF or RM is used.

Taking into account that $\varepsilon(T_{ij}, S) = o_i - ext_i, \forall j$ from (1) and (2), we have:

$$\varepsilon(TS, S, H) = \sum_{i=1}^{n} w_i n_i o_i - \sum_{i=1}^{n} w_i n_i ext_i \quad (6)$$

where n_i is the number of jobs of subtask M'_i (or M_i) occurred in hyperperiod H. The minimization of total weighted error implies the maximization of the second member of the right side hand in (6):

$$\min(\varepsilon(TS, S, H)) \Rightarrow \max \sum_{i=1}^{n} w_i n_i ext_i. \quad (7)$$

From (5) and (7), it follows that this problem of determination of extensions can be reduced to the bounded knapsack problem (BKP) for both scheduling algorithms, EDF and RM.

In BKP a knapsack with weight capacity b and n items, $j = 1, \ldots, n$, with utility values c_j and weights a_j is given. The quantities of items, x_j, have to be calculated with limited total weight of the knapsack, b, and the upper bounds, b_j, maximizing the value of the knapsack's content. BKP can be formulated as [2]:

$$\max \sum_{j=1}^{n} c_j x_j, \text{ subject to } \sum_{j=1}^{n} a_j x_j \leq b, \quad (8)$$

$$0 \leq x_j \leq b_j,\ j = 1, \ldots, n,\ x_j \text{ integer}.$$

Assuming that all weights, a_j, all utility values, c_j, and all upper bounds, b_j, are positive integers, the determination of extensions problem can be reduced to BKP with following parameters:

$$c_j = w_k n_k,\ a_j = n_k,\ x_j = ext_k,\ b_j = o_k, \quad (9)$$

$$b = ext_{MAX},\ j = 1, \ldots, n,\ k = 1, \ldots, n,$$

assuming that the tasks are reordered according to $c_1/a_1 \geq \ldots \geq c_n/a_n$. Since $c_1/a_1 \geq \ldots \geq c_n/a_n$ implies $w_1 \geq \ldots \geq w_n$, which is one of the assumptions, the reordering is not needed, $j = k$. This order is requested in order to apply a branch-and-bound algorithm, Algorithm 1, which is proposed for the calculation of an optimal solution of BKP problem. Algorithm 1 is a modified version of Algorithm Branch-and-Bound Knapsack for the knapsack problem, described in [2]. The modification is done in order to limit the extensions to optional execution times. In Figure 1 a detailed description of Algorithm 1 is given.

```
$L := 0; k := 0;$
REPEAT
    FOR $j := k + 1$ TO $n$ DO
        $x_j := \min\left(\left\lfloor\left(b - \sum_{i=1}^{j-1} a_i x_i\right)/a_j\right\rfloor, o_j\right);$
    ENDFOR
    IF $\sum_{i=1}^{n} c_i x_i > L$ THEN
        $L := \sum_{i=1}^{n} c_i x_i;\ x^* := x;$
    ENDIF
    $l := n - 1;$
    WHILE $l > 0$ DO
        Determine the largest $1 \leq k \leq l$ with $x_k > 0$,
        if does not exist stop;
        $x_k = x_k - 1;$
        IF $\sum_{i=1}^{k} c_i x_i + \frac{c_{k+1}}{a_{k+1}}(b - \sum_{i=1}^{k} a_i x_i) < L + 1$ THEN
                $l := k - 1;$
        ELSE
                EXIT (WHILE);
        ENDIF
    ENDWHILE
UNTIL $l = 0;$
```

Figure 1. Algoritm 1.

The result of Algoritm 1 is an optimal solution, x^*, for BKP, i.e. a set of extensions $(ext_1, \ldots, ext_n)$. For the EDF and RM scheduling algorithms the optimal solutions (extensions) x^*_{EDF} and x^*_{RM} are generated using $ext_{MAX,EDF}$ and $ext_{MAX,RM}$, respectively. Solutions x^*_{EDF} and x^*_{RM} are used for the creation of M' for both scheduling algorithms (M'_{EDF}, M'_{RM}). M'_{EDF} and M'_{RM} are scheduled using EDF and RM instead of scheduling TS. The resulting schedules keep the schedulability of M and minimize the total weighted error of TS in both cases. They are used for scheduling of TS for the EDF and RM scheduling algorithms.

3.2. Two-level scheduling

As mentioned, in this approach task set TS is scheduled on two levels. On the high level, mandatory subset M is scheduled and processor time for optional subtask set O has to be ensured. This is done by adding a new mandatory subtask M_{n+1} to M and as the result a new mandatory subtask set M' is created. M' must not harm the schedulability of M and the execution time of M_{n+1} is assigned to optional subtask set O. The maximum possible execution time for M_{n+1} should be determined, the time intervals assigned to M_{n+1} should be determined and optional subtask set O should be scheduled over found time intervals in a way that the total weighted error is minimized. This should be done for the EDF and RM scheduling algorithms.

Theorem 1. Let τ be a synchronous periodic task set, $\tau = \{T_1, \ldots, T_n\}$, $T_i = (r_i, d_i, e_i, p_i)$, $n, i \in N$, $i \leq n$, $p_i, e_i \in N$, $1 < p_1 \leq \ldots \leq p_n$, $d_i = p_i$, $r_i = 0$, schedulable by EDF and RM with the utilization less than 1, $U(\tau) < 1$. There exists synchronous periodic task set τ', $\tau' = \tau \cup \{T_{n+1}\}$; $T_{n+1} = (e_{n+1}, p_{n+1})$, which is schedulable by EDF and RM with the utilization equal to 1, $U(\tau') = 1$, where $p_{n+1} = lcm(p_1, \ldots, p_n)$ and $e_{n+1} = (1 - U(\tau))lcm(p_1, \ldots, p_n)$.

Proof: Let's define task $T_{n+1} = (r_{n+1}, d_{n+1}, e_{n+1}, p_{n+1})$, where $r_{n+1} = 0, p_{n+1} = H, d_{n+1} = p_{n+1}$ and $e_{n+1} = (1 - U(\tau))H$. It is obvious that task set τ', $\tau' = \tau \cup \{T_{n+1}\}$, is a synchronous periodic task set with $U(\tau') = 1$ and the hyperperiod equal to H, $H = lcm(p_1, \ldots, p_n)$. Task set τ' exists with $U(\tau') = 1$ and therefore it is schedulable by EDF. Let S and S' be the schedules of task sets τ and τ' produced by RM. Both schedules are repeated by H and it is sufficient to analyze the behaviour during H. Task T_{n+1} has the lowest priority in τ' under RM during H. The lowest priority of task T_{n+1} in τ' guarantees the same execution intervals of tasks $T_1, \ldots, T_n$ in S' as in S during H. This implies that each of tasks $T_1, \ldots, T_n$ meets its deadline and that the idle intervals in S can be assigned for the execution of T_{n+1} in S'. The amount of time of all idle intervals in S is equal $(1 - U(\tau))H$ which is equal to e_{n+1} during H. This confirms that task T_{n+1} meets its deadline and proves the schedulability of task set τ' under RM. □

According to Theorem 1 we have $M' = M \cup \{M_{n+1}\}$, $r_{n+1} = 0$, $p_{n+1} = H$, $m_{n+1} = (1 - U(M))H$, $o_{n+1} = 0$, $d_{n+1} = p_{n+1}$, where H is the hyperperiod of M. Resulting subtask set M' is schedulable by EDF and RM with the utilization equal to 1, i.e. the maximum possible execution time is assigned to M_{n+1}.

In order to minimize the total weighted error the processor availability for mandatory subtask M_{n+1}, has to be determined. Let's define a busy period ([1], [7]) as an interval of time in which the processor is never idle. This implies that the processor availability for mandatory subtask M_{n+1}, is equal to the idle periods of the schedule of mandatory subtask set M.

Theorem 2. Let S_1 and S_2 are feasible schedules for task set τ defined in Theorem 1 generated by the EDF and RM scheduling algorithms, respectively. Both schedules have the same busy periods.

Proof: Let's assume that S_1 and S_2 do not have the same busy periods and I, $I = [t_1, t_2)$, be the first time interval during which an idle period is present in one schedule (S_i) and a busy period in the other schedule (S_j). This implies that the amount of time assigned for the execution of the task set is the same in S_i and S_j until t_1. In S_i each started task' job successfully finished its execution (i.e. all deadlines are met) by t_1, while in S_j there is at least one ready job that has to finish its execution. There are two possibilities in S_j at t_1: the execution of a task that just became ready at t_1 and the execution of a task that became ready before t_1. In the first case it is obvious that the task sets scheduled by S_i and S_j are not the same, the task sets do not have the same periods. In the second case in S_i all deadlines are met until t_1 while this is not the case for S_j. Since the same amount of time is assigned to the task set execution in both schedules until t_1 this implies that the execution times are not the same and, further, that the task sets scheduled by S_i and S_j are not the same. This is against the hypothesis. If I is an idle period in one schedule it must be an idle period in another schedule. □

According to Theorem 2 the same algorithm for the determination of idle periods may be applied for feasible schedules generated by EDF and RM. For finding the start and end points of each idle period, $[S_i, E_i)$, $S_i, E_i, i \in N$, $S_i < E_i$, for a feasible schedule of a task set generated by EDF and RM Algorithm 2 is proposed. A detailed description of Algorithm 2 is given in Figure 2.

```
$j := 1;$
$E_0 := 0;$
$r_i := 0; i \leq n$
REPEAT
    $S_{j,0} := 1;$
    $k := 0;$
    REPEAT
        $k := k + 1;$
        $S_{j,k} := \sum_{i=1}^{n} \left\lceil \frac{\max(S_{j,k-1} - r_i, 0)}{p_i} \right\rceil m_i;$
        $res = \min(((E_{j-1} + S_{j,k}) \mod p_i));$
        IF $(S_{j,k} = S_{j,k-1})$ and $(res = 0)$
            $S_{j,k} := S_{j,k} + 1;$
        ENDIF
    UNTIL $(S_{j,k} = S_{j,k-1});$
    $S_j := S_{j,k} + E_{j-1};$
    $E_j := H - \max_{i=1,..,n} \left( \left\lfloor \frac{H - S_j}{p_i} \right\rfloor p_i \right);$
    FOR $i := 1$ TO $n$ DO
        $r_i := (H - E_j) - \left\lfloor \frac{H - E_j}{p_i} \right\rfloor p_i;$
    ENDFOR
    $j := j + 1;$
UNTIL $E_j = H$;
```

Figure 2. Algorithm 2, an iterative algorithm for finding the idle intervals.

As the result of applying Algorithm 2 on M, the available time intervals for scheduling optional subtask O are provided, $TI = \{[S_1, E_1), \ldots, [S_l, E_l)\}$, $l \in N$. Note that precedence constraints between the mandatory and optional subtasks are already included in TI, i.e. there is no need for ready optional tasks to wait with the execution.

Finally, optional subtask set O should be scheduled minimizing the total weighted error of task set TS. This is done by making use of Algorithm WNTU [4]. Algorithm WNTU minimizes the total weighted error. It was developed for an aperiodic task set which implies that the optional subtask set O needs to be translated into aperiodic task set ATS over observed time interval, H. This is done by considering each task's job as a separate task over H. As the result we have aperiodic task set $ATS = \{O_{1,1}, \ldots, O_{1,n1}, O_{2,1}, \ldots, O_{2,n2}, \ldots, O_{n,1}, \ldots, O_{n,nn}\}$, $O_{i,j}$ is the task that corresponds to the jth job of the ith subtask; $O_{ij}(r_{ij}, d_{ij}, e_{ij})$, $r_{ij} = (j-1)p_i$, $d_{ij} = jp_i$, $e_{ij} = o_{ij}$, $n_i = \frac{H}{p_i}$. WNTU algorithm supposes different weights, i.e. $w_1 > \cdots > w_k$. If task set ATS contains more than one task with the same weight, the grouping is needed. All tasks with the same weights form a subset of tasks with the same weights. There are k subsets $(k \leq n)$ each of which contains all jobs of the tasks with the same weight. Let TSS_i be a subset which contains all tasks with weight w_i and V_j the jth busy period block. In addition, let SC be an empty schedule and ET be an empty set. A detailed description of Algorithm WNTU is given in Figure 3.

```
$SC = \emptyset, ET = \emptyset;$
FOR $j := 1$ TO $k$ DO
    $SC_j :=$ Algorithm NTU for $ET \cup TSS_j$;

    Begin (Adjustment Step)
        Let there be $q$ blocks in $SC$: $V_i = [v_{i-1}, v_i]$,
            $1 \leq i \leq q$.
        FOR i := 1 TO q DO
            IF $V_i$ is a task block in $SC$, THEN
                Let task $l$ be executed within $V_i$ in $SC$.
                Let $N(l)$ (resp. $N_j(l)$) be the number of
                time units job $l$ has executed in $SC$ (resp.
                $SC_j$) from the beginning until time $v_i$.
                IF $N(l) > N_j(l)$, THEN
                    assign $(N(l) - N_j(l))$ more time
                    units to task $l$ within $V_i$ in $SC_j$, by
                    replacing any task, except task $l$,
                    that was originally assigned within
                    $V_i$.
                ENDIF
            ENDIF
        ENDFOR
    End

    $SC_j' := SC_j$.
    Set the execution time of each task in TSS$_j$ to be the
    number of nontardy units in $SC_j'$.
    $ET := ET \cup TSS_j$
    $SC :=$ Algorithm NTU for $ET$.
ENDFOR
```

Figure 3. Algorithm WNTU.

In this case Algoritm WNTU makes use of modified Algorithm NTU. Algorithm NTU [4] had to be modified in order to take into account available processor intervals TI. Algorithm NTU schedules an aperiodic task set in decreasing order of tasks' ready times which implies the reordering of a task set that should be scheduled. Same as for Algorithm NTU, the modified version uses $m \times l$ matrix SM to represent a schedule. $SM(i,j)$ contains the number of time units task T_i is scheduled in the jth idle period,

$[S_j, E_j)$. A detailed description of Algorithm NTU is given in Figure 4. l is the number of idle periods and m is the number of tasks to be scheduled.

```
$l_i := E_i - S_i; i \leq l$
FOR $i = 1, \ldots, m$ DO
    Find $\min(p) | S_p > r_i$;
    Find $\max(q) | E_q \leq d_i$;
    IF $p \leq q$
        FOR $j = q, \ldots, p$ DO
            $\delta := \min(l_j, e_i)$;
            $SM(i,j) := \delta; l_j := l_j - \delta; e_i := e_i - \delta$;
        ENDFOR
    ENDIF
ENDFOR
```

Figure 4. Modified Algorithm NTU.

At the end of Algorithm WNTU, the schedule of subtask set O which minimizes the total weighted error of TS is provided. This schedule is used together with schedules of M' generated by EDF and RM.

4. Conclusion

In this paper, the analysis of Imprecise Computation Model for synchronous periodic task sets is provided. The analysis is based on the use of the EDF and RM scheduling algorithms for the mandatory subtask sets and the minimization of total weighted error. The scheduling of the optional subtask set is affected by the minimization of total weighted error. Two different approaches are proposed, one-level and two-level scheduling approach. The results are the optimal schedules with respect to the total weighted error, whether the EDF or RM scheduling algorithm is used for the scheduling of mandatory subtask set.

In one-level scheduling approach, the mandatory subtask set is transformed into the new mandatory subtask set extending the mandatory execution times as much as possible in order to minimize the total weighted error. The scheduling is done for the new mandatory subtask set instead of the original task set on one level using EDF and RM. This approach has simpler implementation, but the flexibility of Imprecise Computation Model is lost. The lower total weighted error is achieved for the EDF scheduling algorithm because of its higher least upper bound to processor utilization with respect to RM.

For two-level scheduling, the scheduling of mandatory subtask set is done on one level using EDF and RM while the scheduling of optional subtask set is done on a separate level minimizing the total weighted error. This approach has more complex implementation, but the flexibility is kept and the time isolation between the mandatory and the optional sets is obtained. The same total weighted error is achieved for the EDF and RM scheduling algorithms because they generate same idle intervals.

In comparison with two-level scheduling, a lower total weighted error is expected for one-level scheduling due to the uniform distribution of job extensions. An interesting application case is the minimization of total error where the weight of error of each task is the same. The result is the maximum processor utilization for the given task sets.

The lowest total weighted error boundary for both scheduling approaches is established, but there are still a lot of different, open questions such as the comparison of two different scheduling approaches, the time complexity, the improvements of existing algorithms, the usage of new algorithms and possible application areas. The findings in the paper are expected to be the theoretical basis for all open questions and improvements in the future.

References

[1] S. BARUAH, R. HOWELL, L. ROSIER, Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor. *Real-Time Systems*, 2 (1990), pp. 301–324.

[2] P. BRUCKER, S. KNUST, *Complex Scheduling*, Berlin Heidelberg: Springer-Verlag; 2006.

[3] W. FENG, J. W. S. LIU, Algorithms for Scheduling Real-Time Tasks with Input Error and End-to-End Deadline. *IEEE Transactions on Software Engineering*, 23(2) (1997), pp. 93–106.

[4] J. Y-T. LEUNG, Imprecise Computation Model: Total Weighted Error and Maximum. In *Handbook of Real-Time and Embedded Systems* (I. Lee, J. Y-T. Leung, S. H. Son, Ed.), (2008) pp. 7-1–7-14. Chapman & Hall/CRC.

[5] C. LIU, J. LAYLAND, Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM* 20(1) (1973), pp. 46–61.

[6] J. W. S. LIU, K. J. LIN, W. K. SHIH, A. C. YU, J. Y. CHUNG, W. ZHAO, Algorithms for Scheduling Imprecise Computation. *IEEE Computer*, 5 (1991), pp. 58–68.

[7] R. PELLIZZONI, G. LIPARI, Feasibility Analysis of Real-Time Periodic Tasks with Offsets. *Real-Time Systems*, 30 (2005), pp. 105–128.

[8] W. K. SHIH, J. W. S. LIU, Algorithms for Scheduling Imprecise Computations with Timing Constraints to Minimize Maximum Error. *IEEE Transactions on Computers*, 44(3) (1995), pp. 466–471.

[9] J. W. S. LIU, K. J. LIN, W. K. SHIH, A. C. YU, J. Y. CHUNG, W. ZHAO, Algorithms for Scheduling Imprecise Computations. In *Foundations of Real-Time Computing: Scheduling and Resource Management* (A. M. Van Tilborg, G. M. Koob, Ed.), (1991) pp. 203–249. Norwell, Kluwer Academic Publishers.

Received: June, 2010
Accepted: November, 2010

Contact addresses:

Damir Poles
Eurocontrol Experimental Centre
Centre de Bois des Bordes BP 15
F-91222 Brétigny sur Orge
France
e-mail: damir.poles@eurocontrol.int

Leo Budin
Faculty of Electrical Engineering and Computing
Unska 3
HR-10000 Zagreb
Croatia
e-mail: leo.budin@fer.hr

DAMIR POLEŠ holds Dipl. Ing. degree in electrical engineering from the Faculty of Electrical Engineering and Computing, University of Zagreb and M.Sc. degree in technical science, Interdisciplinary Postgraduate Study "Guidance and Control of Moving Objects", University of Zagreb. He is a Ph.D. student at the Faculty of Electrical Engineering and Computing, University of Zagreb and he works for the Eurocontrol Experimental Centre as a senior assistant in aircraft performance modelling. His professional interests include real-time systems, modelling and simulations and software engineering.

LEO BUDIN is a Professor Emeritus of computer engineering at the Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia, where he has been lecturing since 1962. He holds the Ph.D., M.Sc. and Dipl. Ing. degree in electrical engineering from the University of Zagreb. His research interest is in the general study of computers and computer systems, along with methodologies in system analysis and design, and computer applications in diverse fields of human activities, within which he has led a number of national research projects. Professor Budin has published more than 100 research and professional papers in journals, conference proceedings and books. He is the author and co-author of several books and he has been the supervisor for a number of Ph.D. and M.Sc. students. Professor Budin has been conferred several awards for significant scientific achievements, and has been both a Humboldt and a Fulbright scholar. He is a fellow of the Croatian Academy of Sciences and Arts and a member of IEEE, IEEE Computer Society, and ACM.