The Problem of Fixed Priority Scheduling with Non-Standard Timing Constraints: Definitions, Examples and Problem Statement

Technical Report at_pstu_ru-rts-2007-01

Maksim Kavalerov
Department of Automation and Telemechanics,
Perm National Research Polytechnic University,
Perm, Russia
mkavalerov@gmail.com

2007

Contents

1 Introduction **3**

2 System Model and General Assumptions **3**

2.1 Hard Real-Time Tasks 4

2.2 On-line Scheduling 5

2.3 Off-line Attribute Assignment 6

2.3.1 Task Attributes and Schedulability 6

2.3.2 Standard Timing Constraint 7

2.3.3 Attribute Assignment for Tasks with Standard Timing Constraints 9

2.4 Non-Standard Timing Constraints 12

2.4.1 General Remarks 12

2.4.2 Examples 12

2.4.2.1 Timing Constraint of a Control Loop Task 12

2.4.2.2 Timing Constraint of an Event Handling Task 14

3 The Baseline Approach to Attribute Assignment Can Be Significantly Improved **16**

4 Conclusion **19**

1 Introduction

Fixed priority scheduling [11] is widely recognized and used for planning and dispatching real-time tasks in various systems such as embedded system, automation control systems, networked control systems. In general, fixed priority scheduling can be characterized as a process consisting of two stages that can be called on-line and off-line. Off-line stage involves schedulability analysis [15], [14] and assignment [1], [13], [12] of attributes to real-time tasks, e.g., priorities, periods, offsets. Schedulability analysis gives the answer whether the task set is schedulable or not if the specified attributes are assigned. A task is called schedulable if it is guaranteed that the timing constraint of the task holds for each task instance, or job. Attribute assignment is also based on timing constraints. Thus timing constraints are crucial for the success rate of scheduling, i.e., for the possibility to find attributes, such as periods and priorities, appropriate for guaranteeing schedulability.

Probably, the most popular timing constraint for real-time tasks is the constraint based on period and deadline. In this paper this constraint is called *standard*. But many original timing constraints, i.e., real-time specifications of the system, are not standard. Maybe the most natural way to schedule tasks with *non-standard* constraints is to make these constraints standard by transforming them. The transformation must produce the resulting, or derived, constraint which is sufficient, i.e., if the derived constraint is met then the original constraint is guaranteed to be met. Then the classical methods of fixed priority scheduling could be applied to setup task attributes. But this approach may lead to over-constraining of the task set [2], making it less probable that the task set is schedulable with the derived constraints.

This paper addresses the issue of scheduling tasks with non-standard constraints, particularly, the problem of how to derive standard timing constraints. First, some notations, definitions and claims are given to form the basis for later considerations of non-standard real-time task scheduling in this paper and future research. Second, some examples of non-standard timing constraints are represented and the conditions of sufficiency for these constraints are derived. These conditions can be used to find sufficient standard timing constraints with such attributes that make the task set schedulable. Finally, an example is presented which demonstrates that the schedulability analysis based on non-standard timing constraints can be very efficient.

2 System Model and General Assumptions

For every time instant there is a real number. The start time of a given real-time system is denoted by 0. Let $t_{\max}$ be the latest observed time instant before the finish time of the real-time system.

We say that the scheduling *time quantum* is the real number ε such that all scheduling events can occur only at time instants $0, \varepsilon, 2\varepsilon, \ldots, k\varepsilon, \ldots$, where k is integer; and some scheduling event can occur at any of these time instants. In particular, the shortest time interval between two consecutive events is equal to ε. Note that for brevity we

will write "α is a multiple of ε" instead of "there exists an integer k such that $\alpha = k\varepsilon$".

All tasks are scheduled on a single processor. All tasks are independent. We assume that the fixed priority preemptive scheduling policy is used.

2.1 Hard Real-Time Tasks

We will denote by $\{\tau_i\}$ the set of all hard real-time tasks in a system. Let n be the number of tasks in $\{\tau_i\}$. Throughout the paper we assume $n \geq 1$. We use $\tau_1, \tau_2, \ldots, \tau_n$ to denote hard real-time tasks. Any task τ_i is either periodic or sporadic, and has a timing constraint. Any task τ_i generates jobs $\{\tau_{i,1}, \tau_{i,2}, \ldots, \tau_{i,j}, \ldots\}$, where $\tau_{i,j}$ is the j-th job of τ_i.

Any job $\tau_{i,j}$ is released to the run queue at its *release time* $r_{i,j}$. The *start time* of execution of job $\tau_{i,j}$ is denoted by $s_{i,j}$. The *finish time* of $\tau_{i,j}$ is denoted by $f_{i,j}$. Clearly, $r_{i,j}$, $s_{i,j}$, $f_{i,j}$ are multiples of ε.

For $\forall j \geq 1$, job $\tau_{i,j+1}$ cannot start before $\tau_{i,j}$ is complete, i.e., before time $f_{i,j}$. Hence, the following condition holds:

$$\text{for } \forall j \geq 1: \quad 0 \leq s_{i,j} < f_{i,j} \leq s_{i,j+1} < f_{i,j+1}\,. \tag{2.1}$$

We assume that $r_{i,0}$, $s_{i,0}$, $f_{i,0}$ are known a priori and satisfy:

$$r_{i,0} \leq s_{i,0} < f_{i,0} \leq r_{i,1}\,. \tag{2.2}$$

If τ_i is periodic, then the following condition holds:

$$\text{for } \forall j \geq 1: \quad r_{i,j} = r_{i,j-1} + T_i = O_i + (j-1)T_i\,, \tag{2.3}$$

where O_i is the *offset* of τ_i and T_i is the *period* of τ_i. We assume that if τ_i is periodic, then $r_{i,0} = O_i - T_i$, and O_i, T_i are multiples of ε.

If τ_i is sporadic, then $O_i = 0$, $r_{i,0} = -T_i$, and the following condition holds:

$$\text{for } \forall j \geq 1: \quad r_{i,j} \geq r_{i,j-1} + T_i\,, \tag{2.4}$$

where T_i is the *minimum period* of τ_i. It is clear that T_i can be easily adjusted to be a multiple of ε. Therefore, we assume that T_i is a multiple of ε if τ_i is sporadic.

It is significant that if τ_i is sporadic, then each $r_{i,j}$ is not known in advance and it is determined by an external event. Therefore (2.4) shows only that there is a minimum interval between two successive jobs. By contrast, if τ_i is periodic then values $\{r_{i,j} | \, j \geq 1\}$ are determined by (2.3), therefore they are known in advance.

The *computation time* of $\tau_{i,j}$, denoted by $C_{i,j}$, is defined to be $f_{i,j} - s_{i,j} - \delta$, where δ is the time when $\tau_{i,j}$ is idling because of its interruption by another job. Clearly, $C_{i,j}$ is strictly greater than zero and it is a multiple of ε. Therefore, the inequality $C_{i,j} \geq \varepsilon$ always holds. We assume that the computation time of any job cannot be affected by the scheduler.

In general, $C_{i,j}$ is not known before time $f_{i,j}$. Let C_i^{Lo} and C_i^{Up} be the lower and upper bounds of $C_{i,j}$, respectively. Then $C_{i,j} \in [C_i^{Lo}, C_i^{Up}]$ for $\forall j \geq 1$. Clearly, we can get C_i^{Lo}, C_i^{Up} before the start of a real-time system, and these bounds can be easily adjusted to be multiples of ε and be greater than zero. Therefore, we assume that C_i^{Lo}, C_i^{Up} are multiples of ε and are greater than zero.

According to fixed priority scheduling policy, each task τ_i has its own priority, denoted by π_i. We assume that all priorities are unique. If $\pi_v < \pi_w$ then the priority of τ_v is said to be higher than the priority of τ_w. Let us remark that π_i is not necessary equal to i.

By $I(v)$ we denote the index of the task with priority equal to value v. Suppose all priorities are integer in the range $[1, n]$; then, for example, $\tau_{I(1)}$ is the highest priority task, $\tau_{I(n)}$ is the lowest priority task, and $\tau_{I(\pi_i-1)}$ is the task immediately preceding τ_i when all tasks are arranged in order of decreasing priority and $\pi_i > 1$.

Thus, each task τ_i has following attributes: the task type property (periodic or sporadic); the timing constraint; and the numbers O_i, T_i, π_i, C_i^{Lo}, C_i^{Up}. Later, we introduce additional attributes.

The fixed priority preemptive scheduling policy is applied to the task set $\{\tau_i\}$. We consider this policy to be represented as a combination of on-line scheduling and off-line attribute assignment. On-line scheduling is required to dispatch jobs at run time; and off-line attribute assignment provides task attribute assignment and schedulability analysis.

2.2 On-line Scheduling

We assume that the scheduler maintains a run queue where released jobs are waiting for their execution. The run queue may contain: jobs of each task of the task set $\{\tau_i\}$; and aperiodic jobs generated by some aperiodic tasks. Each $\tau_{i,j}$ enters the run queue at time $r_{i,j}$, where $r_{i,j}$ is determined either by (2.3) (if τ_i is periodic), or by an external event (if τ_i is sporadic). Each aperiodic job enters the run queue at a time specified by an aperiodic task scheduling algorithm. Each job of the task τ_i has priority π_i. Every aperiodic job is assigned a priority according to the algorithm of aperiodic task scheduling.

The scheduler assigns jobs to the processor according to the fixed priority preemptive policy. In other words, at every time quantum ε the highest priority job in the run queue gains access to the processor if either the processor is idle or a job with a lower priority is currently executed. In the latter case, the job with a lower priority is interrupted and moved back to the run queue. If there is more than one job with the same highest priority in the run queue, then preference is given to the earliest among these jobs.

Figure 2.1 gives an example showing the graphical notation that we will use to denote task execution.

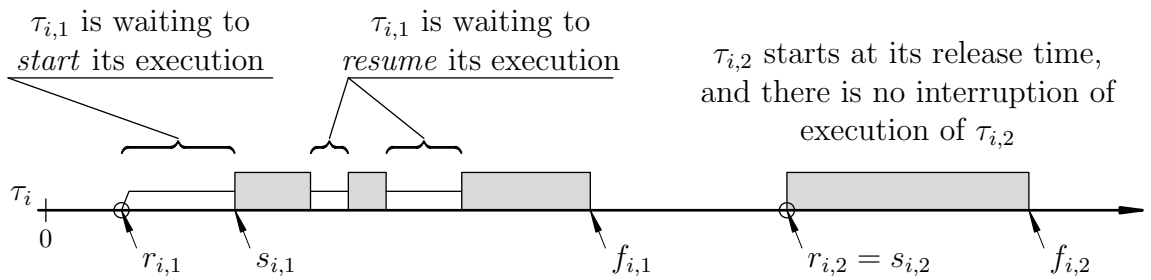

Figure 2.1: An example showing the graphical notation of task execution

2.3 Off-line Attribute Assignment

2.3.1 Task Attributes and Schedulability

It is important to note that during on-line scheduling, timing constraints of tasks $\{\tau_i\}$ are not taken into account explicitly. Indeed, the time when $\tau_{i,j}$ enters the run queue is determined either by (2.3) (i.e., by task attributes O_i, T_i) or by an external event; and jobs generated by tasks $\{\tau_i\}$ are dispatched only according to their priorities. It follows that *only* the set $\{O_i, T_i, \pi_i \mid i \in [1, n]\}$ is taken into account.

Let $\{O_i\}$, $\{T_i\}$, $\{\pi_i\}$ respectively denote the sets $(O_1, O_2, \ldots, O_n)$, $(T_1, T_2, \ldots, T_n)$, $(\pi_1, \pi_2, \ldots, \pi_n)$. Then we say that a tuple $(\{O_i\}, \{T_i\}, \{\pi_i\})$ is assigned to $\{\tau_i\}$ if each task τ_i has offset equal to i-th component of $\{O_i\}$, period equal to i-th component of $\{T_i\}$, and priority equal to i-th component of $\{\pi_i\}$.

Suppose $C_{i,j} = C_i^{Lo} = C_i^{Up}$ for $\forall i \in [1, n]$, $\forall j \geq 1$, and there is no sporadic and aperiodic tasks; then a tuple $(\{O_i\}, \{T_i\}, \{\pi_i\})$ assigned to $\{\tau_i\}$ leads to only one schedule (tasks execution pattern) generated by on-line scheduling. In this case, we say that $(\{O_i\}, \{T_i\}, \{\pi_i\})$ determines (produces) only one *possible* schedule.

In general case, a tuple $(\{O_i\}, \{T_i\}, \{\pi_i\})$ determines a set of possible schedules. Let $Sch(\{O_i\}, \{T_i\}, \{\pi_i\})$ denote such a set. There are different schedules in $Sch(\{O_i\}, \{T_i\}, \{\pi_i\})$ because of variations in $C_{i,j}$ for a task, and because of different arrival patterns of sporadic and aperiodic jobs.

Definition 2.1. Suppose a tuple $(\{O_i\}, \{T_i\}, \{\pi_i\})$ is assigned to $\{\tau_i\}$. Then the timing constraint of a task τ_i is said to be *guaranteed* if this constraint is met for each schedule of $Sch(\{O_i\}, \{T_i\}, \{\pi_i\})$. A task τ_i is called *schedulable* if its timing constraint is guaranteed.

However, we need not only all tasks to be schedulable, but we also need the run queue to be bounded in its size (i.e., to be not overflowed). It is clear that, for arbitrary timing constraints, schedulability of all tasks does not guarantee that the run queue is bounded in its size, and vice versa. Therefore, we introduce the following definition.

Definition 2.2. A task set $\{\tau_i\}$ is called *schedulable* if each task in $\{\tau_i\}$ is schedulable, and the run queue size is upper bounded by a finite number.

We presume that the computation time of a job cannot be affected by the scheduler; therefore we can make the run queue size to be upper bounded only by assigning an appropriate vector $\{T_i\}$. Then, in order to provide the schedulability of $\{\tau_i\}$, it is sufficient to assign appropriate O_i, T_i to each periodic task and to assign an appropriate vector $\{\pi_i\}$ to $\{\tau_i\}$.

The release of each job of a sporadic task τ_i is determined by external events and not by values O_i, T_i (we assume that $O_i = 0$, and T_i is the minimum inter-arrival time of sporadic jobs). But for convenience, we will write "to find a tuple $(\{O_i\}, \{T_i\}, \{\pi_i\})$" presuming that if a task τ_i is sporadic then values O_i, T_i are known a priori and fixed.

Thus, the problem is that we need to find a tuple $(\{O_i\}, \{T_i\}, \{\pi_i\})$ providing the schedulability of $\{\tau_i\}$, or to conclude that there does not exist such a tuple. In general, solving this problem, optimally or through a heuristic, is the main objective for off-line attribute assignment within the fixed priority policy.

The complexity of this problem depends on timing constraints imposed on tasks. Therefore, it is necessary to take into account types of the constraints. Here, we begin with, probably, the most widely used timing constraint.

2.3.2 Standard Timing Constraint

A timing requirement is said to be *standard* if:
(i) release times $r_{i,j}$ are constrained by (2.3) or by (2.4) using O_i and T_i;
(ii) finish times $f_{i,j}$ are constrained by the following condition:

$$\text{for } \forall j \geq 1: \quad f_{i,j} \leq d_{i,j} = r_{i,j} + D_i\,, \tag{2.5}$$

where $d_{i,j}$ is the *deadline* of $\tau_{i,j}$ and D_i is the *relative deadline* of τ_i.

Now, we pass on to standard timing constraints.

If τ_i is sporadic then τ_i is constrained by (2.4). But condtion (2.4) rather describes a property of external events (their minimum inter-arrival time) than imposes a timing constraint. Each release time $r_{i,j}$ of a sporadic task cannot be affected by the scheduler; and (2.4) describes only a specific feature of a given sporadic task representing the attribute T_i of this task. Therefore, we do not consider this inequality as a timing constraint, and we introduce the following definition.

The *standard* timing constraint of a *sporadic* task τ_i is the requirement to satisfy condition (2.5). Clearly, one parameter D_i fully determines standard timing constraint of a sporadic task with the attribute T_i.

We now turn to periodic tasks. Suppose the standard timing constraint of a periodic task is said to be the requirement to satisfy both (2.3) and (2.5); then this definition leads to some inconsistency. Indeed, on the one hand, the standard timing constraint imposes the restriction on $f_{i,j}$ (see (2.5)). Suppose every $f_{i,j}$ is linked to some external activity such as sending a command to an actuator, transmitting a message through a network, etc. Then, as expected, the standard timing constraint imposes the restriction on the behavior of the real-time system. On the other hand, the standard timing constraint imposes the restriction on $r_{i,j}$. If τ_i is periodic, then $r_{i,j}$ is an 'artifact' made by the scheduler using attributes O_i, T_i and according to a scheduling policy. Moreover, in the case of some scheduling policies (e.g., such as a table-driven scheduling), there is no sense in considering $r_{i,j}$ as the time when a job is released to a queue because there is no such a queue. It is clear that the standard timing constraint defined above is attached to some scheduling policies such as policies that assume maintenance of a run-queue. But a timing constraint should appear before and beyond all scheduling policies, and scheduling is only an instrument to meet requirements of the timing constraint.

Therefore, we introduce the following definition.

The *standard* timing constraint of a *periodic* task τ_i is the condition

$$\text{for } \forall j \geq 1: \quad \begin{cases} s_{i,j} \geq O_i^* + (j-1)T_i^* \\ f_{i,j} \leq O_i^* + (j-1)T_i^* + D_i \end{cases}, \tag{2.6}$$

where O_i^*, T_i^* are parameters of the timing constraint such that $O_i^* \geq 0$, $T_i^* \geq 0$.

Notice that O_i^*, T_i^* are not attributes of τ_i, they are parameters of the standard timing constraint. Thus, we see that the constraint (2.6) is not attached to any scheduling policy.

The following claim helps to choose appropriate O_i, T_i to guarantee standard timing constraint (2.6).

Claim 2.1. *If a task τ_i has the standard timing constraint (2.6) and this constraint is met for some O_i, T_i; then this constraint is also met for $O_i = O_i^*$, $T_i = T_i^*$.*

Proof. Suppose constraint (2.6) is met for some O_i, T_i. Then for each $j \geq 1$, one of the following cases is possible: $r_{i,j} < O_i^* + (j-1)T_i^*$; $r_{i,j} > O_i^* + (j-1)T_i^*$; $r_{i,j} = O_i^* + (j-1)T_i^*$.

Consider the first case, i.e., $r_{i,j} < O_i^* + (j-1)T_i^*$ for some j. But $s_{i,j} \geq O_i^* + (j-1)T_i^*$, because constraint (2.6) is supposed to be met. This is possible only if during the time interval $[r_{i,j}, O_i^* + (j-1)T_i^*)$, only higher priority tasks are executed. Clearly, condition (2.6) is also met in the presence of these higher priority tasks if the job $\tau_{i,j}$ is released exactly at the time $O_i^* + (j-1)T_i^*$, i.e., if $r_{i,j} = O_i^* + (j-1)T_i^*$.

Consider the second case, i.e., $r_{i,j} > O_i^* + (j-1)T_i^*$ for some j. Constraint (2.6) is supposed to be met in this case. Then it is clear that constraint (2.6) is also met for this j when $r_{i,j} = O_i^* + (j-1)T_i^*$. Indeed, the first inequality in the system of inequalities in (2.6) is met because $s_{i,j} \geq r_{i,j}$; the second inequality is met because it is met for $r_{i,j} > O_i^* + (j-1)T_i^*$, and because $f_{i,j}$ cannot be increased by a decrease of $r_{i,j}$.

It follows that if the standard timing constraint (2.6) is met for some O_i, T_i, then it is also met when $r_{i,j} = O_i^* + (j-1)T_i^*$ for $\forall j \geq 1$, i.e., when $O_i = O_i^*$, $T_i = T_i^*$ (according to (2.3)). □

Thus, when a task τ_i has standard timing constraint (2.6), it seems to be convenient to assign $O_i = O_i^*$, $T_i = T_i^*$. Let us consider this in more detail.

Suppose $T_i > T_i^*$; then it is impossible to meet condition (2.6) because there exists j such that $O_i + (j-1)T_i > O_i^* + (j-1)T_i^* + D_i$ (i.e., $f_{i,j} > O_i^* + (j-1)T_i^* + D_i$).

Suppose $T_i < T_i^*$ and the constraint (2.6) is met; then it is clear that the run queue size cannot be upper bounded for $\forall j \geq 1$, and the schedulability of $\{\tau_i\}$ is impossible according to Definition 2.2.

It follows that only assignment of $T_i = T_i^*$ can provide the schedulability of $\{\tau_i\}$.

Suppose $O_i < O_i^*$, $T_i = T_i^*$, and the constraint (2.6) is met; then an execution pattern of τ_i with these O_i, T_i is identical to that of τ_i with $O_i = O_i^*$, $T_i = T_i^*$.

Suppose $O_i > O_i^*$, $T_i = T_i^*$, and $O_i = O_i^* + \Delta$. Then this is equivalent to the case when $O_i = O_i^*$, $T_i = T_i^*$ and the constraint (2.6) is adjusted such that its parameter O_i^* is increased by Δ and its parameter D_i is decreased by Δ. Therefore, we may assume that this adjusted constraint is used and $O_i = O_i^*$, $T_i = T_i^*$.

Claim 2.2. *Suppose a periodic task τ_i has standard timing constraint (2.6) and $O_i = O_i^*$, $T_i = T_i^*$; then standard timing constraint is met if and only if condition (2.5) is met.*

Proof. If $O_i = O_i^*$ and $T_i = T_i^*$ then for $\forall j \geq 1$ we have $r_{i,j} = O_i^* + (j-1)T_i^*$. Therefore, the second inequality in (2.6) is equivalent to (2.5). Further, for $\forall j \geq 1$ we have $s_{i,j} \geq r_{i,j}$. Therefore, if $O_i = O_i^*$ and $T_i = T_i^*$ then the first inequality in the system in (2.6) is always true. □

This explains why the attributes O_i, T_i are often considered as parameters of a timing constraint, i.e., condition (2.5) is often considered as a standard timing constraint. But for a generalized approach to timing constraints, we need to distinguish the attributes O_i, T_i from parameters of the timing constraint of this task. This enables to consider standard timing constraints as a special case of more generalized timing constraints.

From now on we assume that if a periodic task τ_i has the standard timing constraint (2.6) then $O_i = O_i^*$, $T_i = T_i^*$.

Thus for periodic tasks, the above-mentioned standard timing requirement expressed with the conditions (2.3) and (2.5) can be considered as a manifestation of the standard timing constraint (2.6).

2.3.3 Attribute Assignment for Tasks with Standard Timing Constraints

We say that a tuple $(\{O_i\}, \{T_i\}, \{\pi_i\})$ is *admissible* if $\{\tau_i\}$ is schedulable when this tuple is assigned to $\{\tau_i\}$.

Ideally, during attribute assignment an admissible tuple $(\{O_i\}, \{T_i\}, \{\pi_i\})$ is produced if and only if there exists such a tuple. Clearly, in the case of arbitrary timing constraints, this is impossible in practice, i.e., the attribute assignment algorithm cannot find an admissible tuple in acceptable time.

But search for an admissible tuple is much easier if $\{\tau_i\}$ consists of tasks with only standard timing constraints and $O_i = O_i^*$, $T_i = T_i^*$ for each τ_i.

Claim 2.3. *Suppose all timing constraints of $\{\tau_i\}$ are standard. Then the run queue size is bounded, if all tasks in $\{\tau_i\}$ are schedulable and each periodic task τ_i has attributes $O_i = O_i^*$, $T_i = T_i^*$.*

> *Proof.* Suppose all tasks in $\{\tau_i\}$ are schedulable. Then for each sporadic task, its constraint (2.5) is met. Therefore, the time when a sporadic job $\tau_{i,j}$ is in the run queue is bounded by D_i. Then the maximum number of sporadic jobs in the run queue is bounded, taking into account (2.4).
>
> If for each periodic task: (i) its constraint (2.6) is met, and (ii) $O_i = O_i^*$, $T_i = T_i^*$; then $r_{i,j} = O_i^* + (j-1)T_i^*$ and $f_{i,j} < O_i^* + (j-1)T_i^* + D_i = r_{i,j} + D_i$. Therefore, the time when a periodic job $\tau_{i,j}$ is in the run queue is bounded by D_i. Then the maximum number of periodic jobs in the run queue is bounded, taking into account condition (2.3). □

Suppose all timing constraints of $\{\tau_i\}$ are standard; then taking into account Claims 2.1 and 2.3, we conclude that if an admissible tuple $(\{O_i\}, \{T_i\}, \{\pi_i\})$ exists, then there exists an admissible tuple $(\{O_i\}, \{T_i\}, \{\pi_i\})$ such that $O_i = O_i^*$, $T_i = T_i^*$ for each τ_i. Thus, the search for an admissible tuple $(\{O_i\}, \{T_i\}, \{\pi_i\})$ is reduced to the search of a set $\{\pi_i\}$.

A priority assignment algorithm searches for a set $\{\pi_i\}$ that provides the schedulability of $\{\tau_i\}$. A priority assignment algorithm is called *optimal* if it is unable to find $\{\pi_i\}$ only when there are no sets $\{\pi_i\}$ that provide the schedulability of $\{\tau_i\}$.

In his paper [1], Audsley proposed a priority assignment algorithm. This algorithm is optimal if all timing constraints of $\{\tau_i\}$ are standard. Of course, different descriptions of this algorithm are possible. For example, see a description of the algorithm presented in [9, Section 7].

We introduce a new description of this algorithm, because we need to make a more general interface to the algorithm and to maintain the uniform approach to algorithm descriptions throughout the paper.

It is important to note that we use a Pascal-like pseudo code to describe each algorithm in the paper. In addition to the Pascal control structures, this pseudo code uses

the **return** statement. The **return** statement terminates execution of the function, returns the value of the expression that follows the statement, and transfers control to the expression that called the function.

We use statements with **goto** instead of statements with **break** or **continue** when this makes the pseudo code clearer.

Keywords of the pseudo code are in **bold** font. Variables are in *italic* font. Names of procedures and functions are in *slanted sans serif* font.

We assume that the task set $\{\tau_i\}$ and the variable n are defined globally. Therefore all procedures and functions can access n as well as all tasks in $\{\tau_i\}$ and all attributes of these tasks.

Thus, the above-mentioned algorithm is described in Listing 2.1 as a function that takes no arguments and returns a Boolean value indicating the result of the assignment (see comments at lines 16, 19).

Listing 2.1: Priority assignment according to Algorithm 1 in [1]

```
1  function PriorityAssignment_Audsley: Boolean;
2  label
3      NextPriorityLevel;
4  var
5      v:Integer;                                   {current priority level}
6      w:Integer;          {initial priority of a candidate task for priority level v}
7  begin
8      for w := 1 to n do π_w := w;                 {assign initial arbitrary priorities}
9      for v := n downto 1 do          {for all priority levels starting with the lowest}
10     begin
11         for w := v downto 1 do   {check all candidate tasks for the priority level}
12         begin
13             if w <> v then swap priorities of τ_I(w) and τ_I(v);
14             if Schedulable(τ_I(v)) then goto NextPriorityLevel;
15         end;
16         return False;                        {the task set {τ_i} is NOT schedulable}
17         NextPriorityLevel:
18     end;
19     return True;        {set {π_i} is assigned and all tasks in {τ_i} are schedulable}
20 end;
```

According to the definition of $I(v)$ (see Section 2.1), the notations $\tau_{I(w)}$ and $\tau_{I(v)}$ stand for the tasks with priority w and v respectively. At line 13 a new candidate task for priority level v (task $\tau_{I(w)}$) exchanges its priority with the previous candidate task (task $\tau_{I(v)}$). Further, at line 14 this new candidate task is denoted by $\tau_{I(v)}$, because it assigned priority v as a result of the execution of line 13. Thus the same candidate task is denoted by $\tau_{I(w)}$, at line 13, and by $\tau_{I(v)}$, at line 14.

Clearly, it is necessary to assign $\{O_i\}$, $\{T_i\}$ to $\{\tau_i\}$ before the function *PriorityAssignment_Audsley* is called.

We need to define the function *Schedulable*(τ_i). But first let us introduce the following definition.

Definition 2.3. The *schedulability condition* of task τ_i is a sufficient condition for τ_i to be schedulable. We stress that a schedulability condition is not always a necessary and sufficient condition but always a sufficient condition.

Suppose that: (i) the timing constraint of task τ_i is standard, and (ii) τ_i is either periodic with $O_i = O_i^*$, $T_i = T_i^*$ or sporadic. Then using (2.5) and Claim 2.2, it is clear that the condition

$$Up(f_{i,j} - r_{i,j}) \leq D_i\,, \tag{2.7}$$

is the schedulability condition for τ_i, where $Up(f_{i,j} - r_{i,j})$ denotes an upper bound of $f_{i,j} - r_{i,j}$ for $\forall j \geq 1$. The value $Up(f_{i,j} - r_{i,j})$ is often called the *response time*. Clearly, if $Up(f_{i,j} - r_{i,j})$ is the maximum of $f_{i,j} - r_{i,j}$ for $\forall j \geq 1$ then the schedulability condition (2.7) is necessary and sufficient. There are several approaches to calculate $Up(f_{i,j} - r_{i,j})$, e.g., see [3, 4, 8–10].

The first version of the function *Schedulable*(τ_i) is defined in Listing 2.2.

Listing 2.2: Analysis of schedulability of task τ_i with the standard timing constraint

```
1  function Schedulable(τ_i): Boolean;
2  begin
3      Calculate Up(f_{i,j} − r_{i,j}) using the attributes of τ_i and the attributes of higher
       priority tasks;
4      return the boolean result of condition (2.7);
5  end;
```

Finally, the attribute assignment algorithm for tasks with only standard timing constraints is presented in Listing 2.3.

Listing 2.3: Attribute assignment for tasks with only standard timing constraints

```
1  function AttributeAssignment_StandardTC: Boolean;
2  var
3      i:Integer;
4  begin
5      for i := 1 to n do                                   {Assign {O_i} and {T_i}}
6      begin
7          if τ_i is periodic then begin O_i := O_i^*; T_i := T_i^*; end;
8      end;
9      return PriorityAssignment_Audsley;                   {Assign {π_i}}
10 end;
```

Claim 2.4. *Suppose all timing constraints of* $\{\tau_i\}$ *are standard and the function AttributeAssignment_StandardTC returns* True*, then a tuple* $(\{O_i\}, \{T_i\}, \{\pi_i\})$ *is assigned to* $\{\tau_i\}$ *and the task set* $\{\tau_i\}$ *is schedulable.*

Proof. Suppose all timing constraints of $\{\tau_i\}$ are standard.

The **for** statement in Listing 2.3 assigns $\{O_i\}$ and $\{T_i\}$ to $\{\tau_i\}$. Line 9 assigns $\{\pi_i\}$ to $\{\tau_i\}$ if the function *PriorityAssignment_Audsley* returns True. Therefore a tuple ($\{O_i\}$, $\{T_i\}$, $\{\pi_i\}$) is assigned to $\{\tau_i\}$ if the function *AttributeAssignment_StandardTC* returns True.

If the function *AttributeAssignment_StandardTC* returns True then the function *PriorityAssignment_Audsley* returns True and all tasks in $\{\tau_i\}$ are schedulable. Taking into account Claim 2.3 and Line 7 in Listing 2.3, it is clear that the run queue size is bounded. Thus, according to Definition 2.2, the task set $\{\tau_i\}$ is schedulable if the function *AttributeAssignment_StandardTC* returns True. □

2.4 Non-Standard Timing Constraints

2.4.1 General Remarks

A timing constraint is called *original* if it is presented as a timing requirement in the system specification.

Definition 2.4. A timing constraint α is called *sufficient* for a timing constraint β if (α is met)$\Rightarrow$(β is met).

A sufficient timing constraint can be derived from an original timing constraint during system design.

A timing constraint for task τ_i is called *non-standard* if it cannot be expressed exactly by a standard timing constraint (i.e., by (2.5) when τ_i is sporadic or by (2.6) when τ_i is periodic).

Let us consider how to schedule a task when the *fixed priority scheduling* policy is used. Suppose each task in $\{\tau_i\}$ has either standard timing constraint (2.6) or a non-standard timing constraint.

Then the *baseline approach* to fixed priority scheduling for non-standard timing constraints is described as follows:
(i) for each task with non-standard timing constraint, derive a sufficient standard timing constraint for respective non-standard timing constraint (as a result, each task in $\{\tau_i\}$ will have the standard timing constraint (2.6));
(ii) use function *AttributeAssignment_StandardTC* to assign attributes to tasks in $\{\tau_i\}$.

The baseline approach can be considered as the simplest way to deal with non-standard timing constraints in fixed priority scheduling.

2.4.2 Examples

2.4.2.1 Timing Constraint of a Control Loop Task

Let us consider a closed-loop control system and the corresponding hard real-time task τ_i. We assume that the execution interval $[s_{i,j}, f_{i,j}]$ of each job $\tau_{i,j}$ contains one *sampling instant* when sensor data are acquired and available for τ_i and one *actuation instant* when τ_i sends a command to the actuator. Let $x_{i,j}$ denote the sampling instant of job $\tau_{i,j}$. Let $y_{i,j}$ denote the actuation instant of job $\tau_{i,j}$.

The time $x_{i,j} - x_{i,j-1}$ (i.e., the time between two consecutive sampling instants for job $\tau_{i,j}$) is called the *sampling interval* of $\tau_{i,j}$ and is denoted by $Txx_{i,j}$. Ideally, all sampling intervals should be equal to each other. But in reality, sampling intervals are not

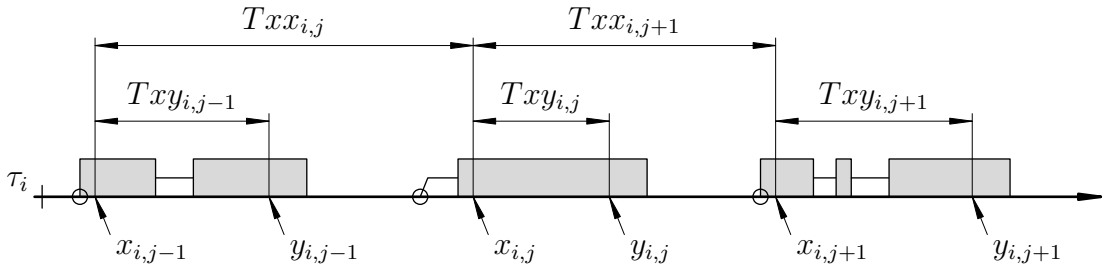

Figure 2.2: Sampling intervals and sampling-actuation delays of a control loop task

constant and they may vary from each other (see Figure 2.2). At least, they should be bounded by a given interval. In other words, the condition $Txx_{i,j} \in [Txx_i^{\min}, Txx_i^{\max}]$ should hold for $\forall j > 1$, where $Txx_i^{\min}$, $Txx_i^{\max}$ are predefined constants.

The time $y_{i,j} - x_{i,j}$ (i.e., the time between sampling instant and actuation instant of job $\tau_{i,j}$) is called the *sampling-actuation delay* of $\tau_{i,j}$ and is denoted by $Txy_{i,j}$. In general, sampling-actuation delay are not constant and they may vary from each other (see Figure 2.2). At least, they should be upper bounded, i.e., the condition $Txy_{i,j} \leq Txy_i^{\max}$ should hold for $\forall j \geq 1$, where $Txy_i^{\max}$ is a predefined constant.

For simplicity, here we assume that $x_{i,j} = s_{i,j}$, $y_{i,j} = f_{i,j}$ for $\forall j \geq 1$. This assumption is commonly used, but it can be relaxed.

Using the above mentioned assumption, an original timing constraint for a given control loop task can be expressed as the condition

$$\text{for } \forall j \geq 1: \quad \begin{cases} Txx_i^{\min} \leq s_{i,j} - s_{i,j-1} \leq Txx_i^{\max} \\ f_{i,j} - s_{i,j} \leq Txy_i^{\max} \end{cases}, \tag{2.8}$$

and the value $s_{i,0}$ is either a constant or a variable. If $s_{i,0}$ is a constant then it is one of the parameters of the constraint (2.8) (the other parameters are: $Txx_i^{\min}$, $Txx_i^{\max}$, $Txy_i^{\max}$). If $s_{i,0}$ is a variable then obviously an arbitrary offset O_i is allowed for task τ_i with the constraint (2.8). Further, if the value of O_i should be upper bounded then variable $s_{i,0}$ should be upper bounded too.

Obviously, timing constraint (2.8) is non-standard.

This constraint is quite similar to the flexible timing constraint defined in Section 5.2 in [6], but constraint (2.8) is for a single task (not for two tasks: the sampling and actuation tasks). Also, constraint (2.8) is quite similar to the flexible timing constraint defined in Section 1.1 in [5], but intervals $[Txx_i^{\min}, Txx_i^{\max}]$, $[0, Txy_i^{\max}]$ are used instead of sets FH, FT. Notice that these intervals can be considered as sets FH, FT because the time quantum ε is used and therefore each interval is a finite set of values.

Constraint (2.8) is obtained during the design stage of the control loop. We assume that satisfactory control performance for the control loop is achieved if this constraint is guaranteed.

It is clear that the longer intervals $[Txx_i^{\min}, Txx_i^{\max}]$, $[0, Txy_i^{\max}]$ facilitate schedulability of τ_i. Therefore during the design stage, these intervals should be maximized, while providing satisfactory control performance. In particular, the intervals can be significantly increased by using the compensation approach [6] and the stability analysis [7].

To use the basic approach to fixed priority scheduling (described in Section 2.4.1) we should be able to derive a sufficient standard timing constraint for the constraint (2.8).

The following claim helps us to solve this problem.

Claim 2.5. *The standard timing constraint (2.6) is sufficient for the non-standard timing constraint (2.8) if the following condition holds:*

$$\begin{cases} O_i^* \geq s_{i,0} + Txx_i^{\min} \\ O_i^* \leq s_{i,0} + Txx_i^{\max} - D_i + C_i^{Lo} \\ T_i^* \geq Txx_i^{\min} + D_i - C_i^{Lo} \\ T_i^* \leq Txx_i^{\max} - D_i + C_i^{Lo} \\ D_i \leq Txy_i^{\max} \end{cases} . \tag{2.9}$$

Proof. According to Definition 2.4, we have to prove that if condition (2.9) holds then ((2.6) is met)⇒((2.8) is met) for $\forall j \geq 1$.

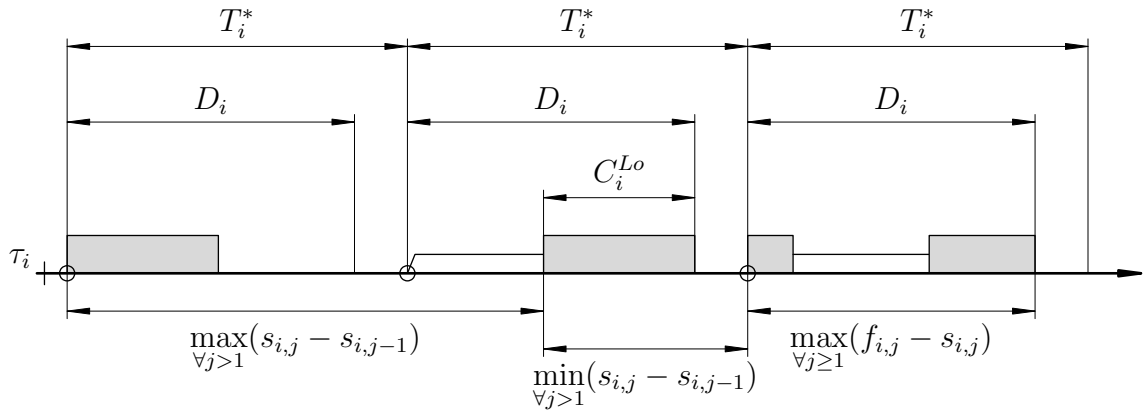

Figure 2.3: The bounds for $s_{i,j} - s_{i,j-1}$ and $f_{i,j} - s_{i,j}$ if the standard timing constraint (2.6) is met

If (2.6) is met then for $\forall j > 1$ the following conditions hold: $s_{i,j} - s_{i,j-1} \geq T_i^* - D_i + C_i^{Lo}$; $s_{i,j} - s_{i,j-1} \leq T_i^* + D_i - C_i^{Lo}$; $f_{i,j} - s_{i,j} \leq D_i$ (see Figure 2.3). Further, if condition (2.9) holds then: $T_i^* - D_i + C_i^{Lo} \geq Txx_i^{\min}$; $T_i^* + D_i - C_i^{Lo} \leq Txx_i^{\max}$; $D_i \leq Txy_i^{\max}$, and consequently: $s_{i,j} - s_{i,j-1} \geq Txx_i^{\min}$; $s_{i,j} - s_{i,j-1} \leq Txx_i^{\max}$; $f_{i,j} - s_{i,j} \leq Txy_i^{\max}$. Thus, if condition (2.9) holds then ((2.6) is met)⇒((2.8) is met) for $\forall j > 1$. It remains to prove that if (2.9) holds then ((2.6) is met)⇒((2.8) is met) for $j = 1$.

If (2.6) is met then: $s_{i,1} \geq O_i^*$; $s_{i,1} \leq O_i^* + D_i - C_i^{Lo}$; $f_{i,1} - s_{i,1} \leq D_i$. Further, if condition (2.9) holds then: $O_i^* \geq s_{i,0} + Txx_i^{\min}$; $O_i^* + D_i - C_i^{Lo} \leq s_{i,0} + Txx_i^{\max}$; $D_i \leq Txy_i^{\max}$, and consequently: $s_{i,1} - s_{i,0} \geq Txx_i^{\min}$; $s_{i,1} - s_{i,0} \leq Txx_i^{\max}$; $f_{i,1} - s_{i,1} \leq Txy_i^{\max}$. Thus, if condition (2.9) holds then ((2.6) is met)⇒((2.8) is met) for $j = 1$. □

2.4.2.2 Timing Constraint of an Event Handling Task

Let us assume that an event handling task τ_i has to react to external events. Let $x_{i,j}$ denote a *detection instant* of job $\tau_{i,j}$ when the information about new external events is acquired. We assume that this information can be about all events occurred after $x_{i,j-1}$. Let $y_{i,j}$ denote a *reaction instant* of job $\tau_{i,j}$ when τ_i reacts to the detected events by sending a command (e.g., to an actuator). We assume that the execution interval $[s_{i,j}, f_{i,j}]$ of each job $\tau_{i,j}$ contains one detection instant $x_{i,j}$ and one reaction instant $y_{i,j}$. It is important that after each detection instant, task τ_i has to react to each detected

event within a given time interval with length $Txxy_i^{\max}$ (the reason for this notation will become clear below). Thus for any handled event, the following condition holds: $t_e \le x_{i,j} < y_{i,j} \le (t_e + Txxy_i^{\max})$, where t_e is the time instant when the event occurs, $x_{i,j}$ is the time instant when this event is detected, $y_{i,j}$ is the time instant when τ_i reacts to this event.

We now give an example of an event handling task. Task τ_i has to control temperature of an object using lower and upper thresholds: α, β. At each detection instant, the existent temperature is measured. At each reaction instant, a command is issued to preserve the temperature within the range $[\alpha, \beta]$ or to bring the temperature back to this range. We assume that a handled event is that the temperature gets outside the range $[\alpha, \beta]$. The information about this event can be available at the next detection instant if the temperature is always outside the range until this instant. For any handled event, the control constraint is described as follows: within the time interval $[t_e, t_e + Txxy_i^{\max}]$ (the event occurs at time t_e), either the temperature comes back to the range $[\alpha, \beta]$ or a command is issued to bring the temperature back to this range.

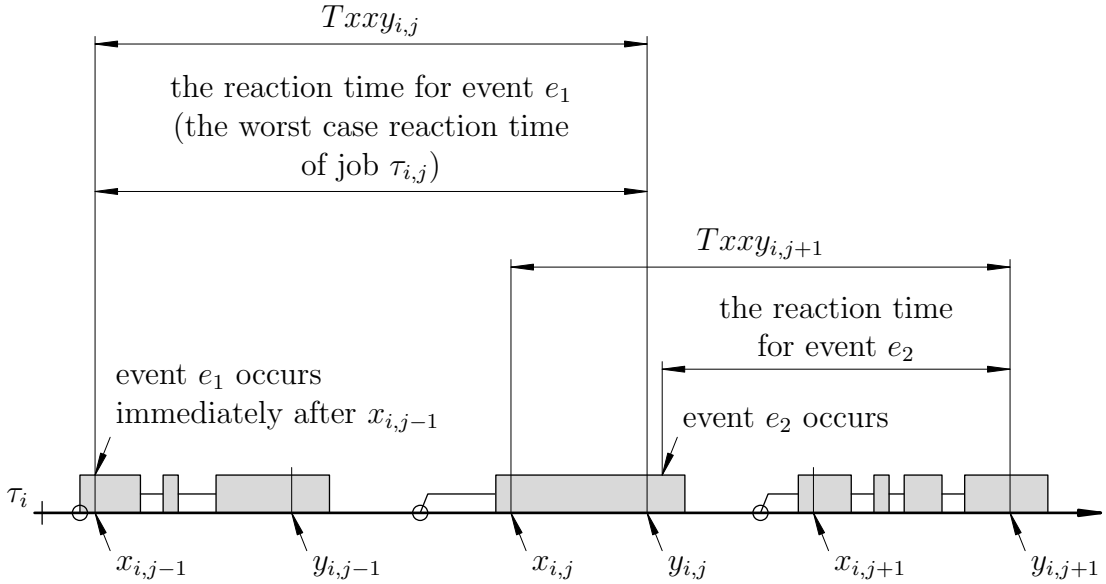

Figure 2.4: Reaction times of an event handling task

For $\forall j > 1$, in the worst case, an event occurs immediately after a detection instant $x_{i,j-1}$, i.e., at the instant $x_{i,j-1} + \gamma$, where γ tends to 0 from above. Then the event is detected only after $x_{i,j}$ (i.e., after the *next* detection instant), if this event is still relevant, e.g., if the temperature is still outside the specified range. Therefore, task τ_i can react to this event only at instant $y_{i,j}$ (i.e., after the *next* detection instant). Thus, the worst case *reaction time* of job $\tau_{i,j}$ is equal to $y_{i,j} - x_{i,j-1}$, i.e., is equal to the time interval beginning at $x_{i,j-1}$, containing $x_{i,j}$, and ending at $y_{i,j}$ (see Figure 2.4). And this interval will be denoted by $Txxy_{i,j}$.

Clearly, the condition $Txxy_{i,j} \le Txxy_i^{\max}$ should hold for $\forall j > 1$, where $Txxy_i^{\max}$ is a predefined constant mentioned above.

Here we use the same assumption as in Section 2.4.2.1, i.e., $x_{i,j} = s_{i,j}$, $y_{i,j} = f_{i,j}$ for $\forall j \ge 1$. Thus, an original timing constraint for a given event handling task can be expressed as the condition

$$\text{for } \forall j \ge 1: \quad f_{i,j} - s_{i,j-1} \le Txxy_i^{\max}, \tag{2.10}$$

and the value $s_{i,0}$ is either a constant or a variable. If $s_{i,0}$ is a constant then it is one of the parameters of the constraint (2.10) (the other parameter is $Txxy_i^{\max}$). If $s_{i,0}$ is a variable then obviously an arbitrary offset O_i is allowed for task τ_i with the constraint (2.10). Further, if the value of O_i should be upper bounded then variable $s_{i,0}$ should be upper bounded too.

Obviously, the timing constraint (2.10) is non-standard.

This constraint is quite similar to the timing constraint defined in Section 3 in [2], but the constraint (2.10) does not impose a restriction on the value $s_{i,j} - s_{i,j-1}$.

As in the case of the constraint (2.8) in Section 2.4.2.1, the following claim can be used to derive a sufficient standard timing constraint for each original constraint (2.10).

Claim 2.6. *The standard timing constraint (2.6) is sufficient for the non-standard timing constraint (2.10) if the following condition holds:*

$$\begin{cases} O_i^* + D_i \leq s_{i,0} + Txxy_i^{\max} \\ T_i^* + D_i \leq Txxy_i^{\max} \end{cases} . \tag{2.11}$$

Proof. According to Definition 2.4, we have to prove that if condition (2.11) holds then ((2.6) is met)⇒((2.10) is met) for $\forall j \geq 1$.

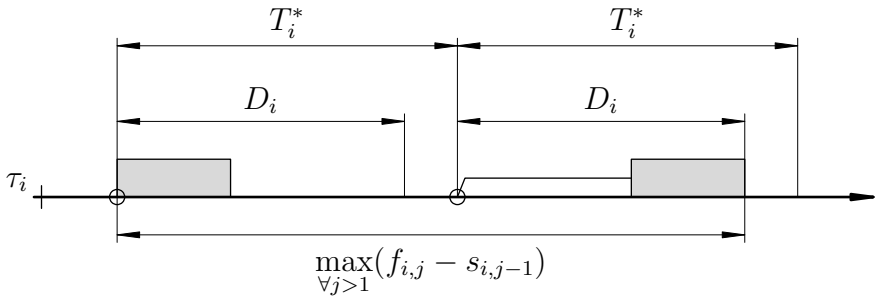

Figure 2.5: The upper bound for $f_{i,j} - s_{i,j-1}$ if the standard timing constraint (2.6) is met

If condition (2.6) is met then the condition $f_{i,j} - s_{i,j-1} \leq T_i^* + D_i$ holds for $\forall j > 1$. Further, if condition (2.11) holds then: $T_i^* + D_i \leq Txxy_i^{\max}$, and consequently $f_{i,j} - s_{i,j-1} \leq Txxy_i^{\max}$. Thus, if condition (2.11) holds then ((2.6) is met)⇒((2.10) is met) for $\forall j > 1$. It remains to prove that if (2.11) holds then ((2.6) is met)⇒((2.10) is met) for $j = 1$.

If (2.6) is met then $f_{i,1} \leq O_i^* + D_i$. Further, if condition (2.11) holds then: $O_i^* + D_i \leq s_{i,0} + Txxy_i^{\max}$, and consequently: $f_{i,1} - s_{i,0} \leq Txxy_i^{\max}$. Thus, if condition (2.11) holds then ((2.6) is met)⇒((2.10) is met) for $j = 1$. □

3 The Baseline Approach to Attribute Assignment Can Be Significantly Improved

The drawback of the baseline approach to attribute assignment is that the transition from the original non-standard timing constraint to the sufficient standard timing constraint leads to overconstraining [2]. In other words, the schedulability analysis, based

on standard timing constraints, could be too pessimistic, i.e., it cannot guarantee that $\{\tau_i\}$ is schedulable when in the same conditions the analysis, based on original non-standard timing constraint, guarantees that $\{\tau_i\}$ is schedulable. Let us explain this problem using the following example.

Let τ_i be a task with non-standard timing constraint 2.8, and $C_{i,j} = C_i^{Lo} = C_i^{Up}$ for $\forall j \geq 1$. Suppose that a sufficient standard timing constraint is chosen such that condition 2.9 and the attributes of τ_i are set as follows $O_i = O_i^*$, $T_i = T_i^*$. Also, suppose that the following settings are established $D_i = Txy_i^{\max}$, $Txx_i^{\max} = T_i + D_i - C_i^{Lo}$, $Txx_i^{\min} = T_i - D_i + C_i^{Lo}$, and they are shown in Figure 3.1.

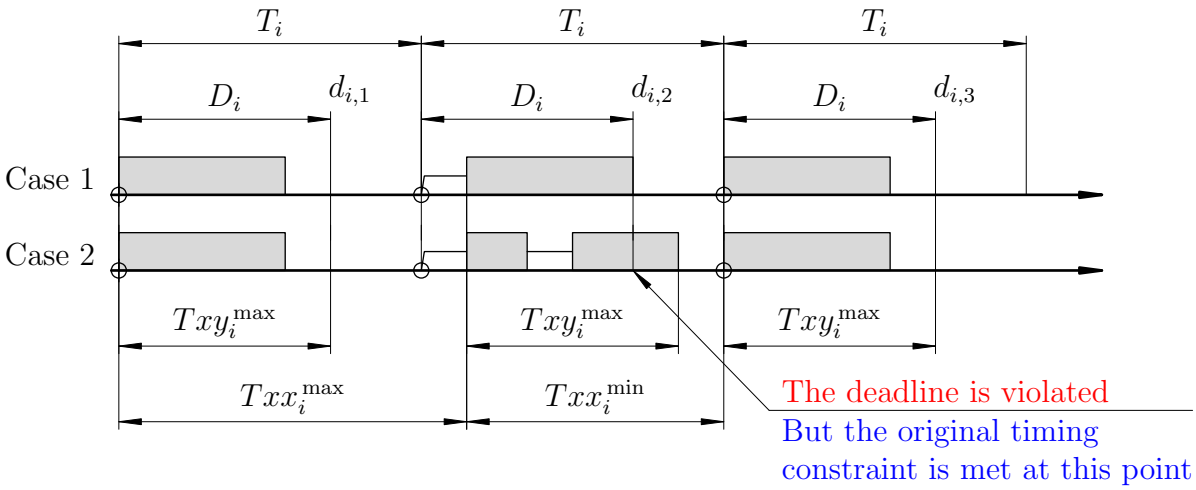

Figure 3.1: Two cases of execution of the same task τ_i with timing constraint 2.8 and the corresponding sufficient timing constraint

Two cases of task τ_i execution are represented and only three first jobs $\tau_{i,1}$, $\tau_{i,2}$, $\tau_{i,3}$ are considered.

In Case 1 the start of $\tau_{i,2}$ is delayed because of some higher priority jobs not shown here. But the deadlines are met for the jobs, indeed, the finish time of each job is not greater then the absolute deadline. Clearly, the original timing constraint is also met as it can be easily seen in Figure 3.1.

Case 2 differs only in that job $\tau_{i,2}$ is interrupted during its execution. This leads to violation of the absolute deadline $d_{i,2}$, i.e., violation of the sufficient standard timing constraint. But the original timing constraint is met because $\tau_{i,2}$ does not go beyond the corresponding interval labeled with $Txy_i^{\max}$. This case highlights the problem with sufficient standard timing constraint that is derived from the original non-standard constraint. This derivation causes the lost of some information about the original constraint making the derived constraint more pessimistic and overconstraining. But only the original non-standard constraint is included in the specification of the system, therefore only this constraint must be met, and violation of the deadline shown in Figure 3.1 does not violate the specification.

Clearly, following the above example, we may conclude that in many cases the average success rate of a scheduling algorithm can be increased by using original non-standard timing constraint during schedulability analysis. In average, the increase in success rate leads to the decrease in spending for hardware and software optimization.

The following example shows how the schedulability analysis based on the original timing constraints can lead to the decrease in spending.

Consider the task set $\{\tau_i\} = \{\tau_1, \tau_2\}$ and $\varepsilon = 1$. Task τ_1 is sporadic with $T_1 = 10$, its timing constraint is determined by the attribute $D_1 = 5$. Task τ_2 is periodic with the original non-standard timing constraint 2.8 having the following attributes: $Txx_2^{\min} = 50$, $Txx_2^{\max} = 60$, $Txy_2^{\max} = 60$, and $s_{2,0} = -55$. Obviously, T_i is not known.

Suppose $C_1^{Lo} = C_1^{Up} = 5$, $C_2^{Lo} = C_2^{Up} = 25$; then τ_i must have the highest priority to be schedulable because $C_1^{Lo} = D_1$. Therefore $\pi_1 = 1$, $\pi_2 = 2$, i.e., all priorities are assigned.

According to the baseline approach 2.4.1, the sufficient standard timing constraint is needed. In general, there exist many different tuples (O_i^*, T_i^*, D_i) such that each of them determines a sufficient standard timing constraint. It can be easily checked that the best tuple is determined by $O_2^* = 0, T_2^* = 55, D_2 = 30$. But even in this case some deadlines are violated as can be found in Figure 3.2.

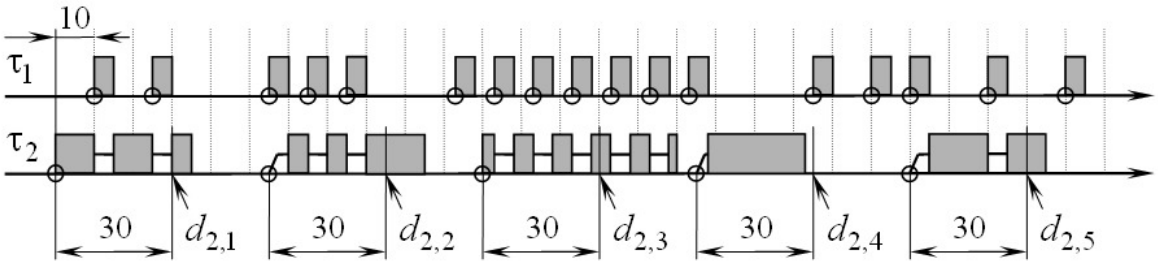

Figure 3.2: Some deadlines are violated for jobs with 100% WCET, i.e., with original hardware performance

We try to reduce worst-case execution time (WCET) of each job. In many cases, this can be done just by increasing the computing power or the hardware performance. For simplicity, suppose that WCETs are decreased according to the following equation: $WCET_{new} = WCET_{old}\frac{P_{old}}{P_{new}}$, where P_{old}, P_{new} are old and new hardware performance.

Figure 3.3, 3.4 show that even with decreased WCETs, namely when WCET is equal 80% and 60% of the original WCETs, some deadlines are violated anyway.

Notice that if WCET decreases then C_i^{Lo} decreases, in turn, according to 2.9, this leads to the decrease of D_i.

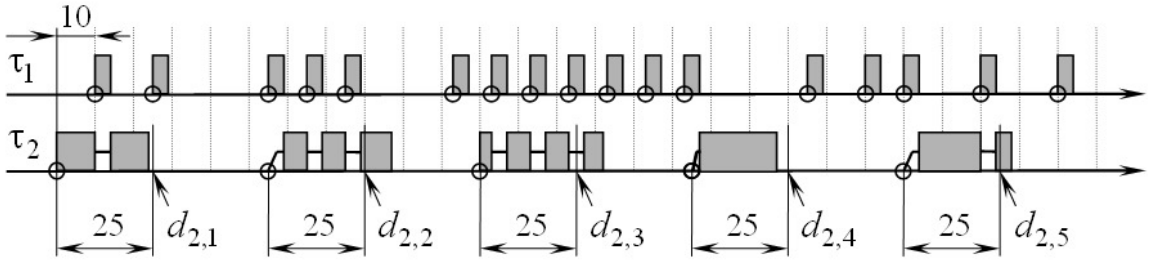

Figure 3.3: Some deadlines are violated for jobs with 80% WCET, i.e., when the hardware performance is increased roughly by 25%

Only with 40% WCET all deadlines are met, but this can be achieved, under our assumptions, only when the hardware performance is increased roughly by 150%, see Figure 3.5.

It is remarkable that the original non-standard timing constraint of τ_2 is met even with the original hardware performance, i.e., without the additional cost needed to

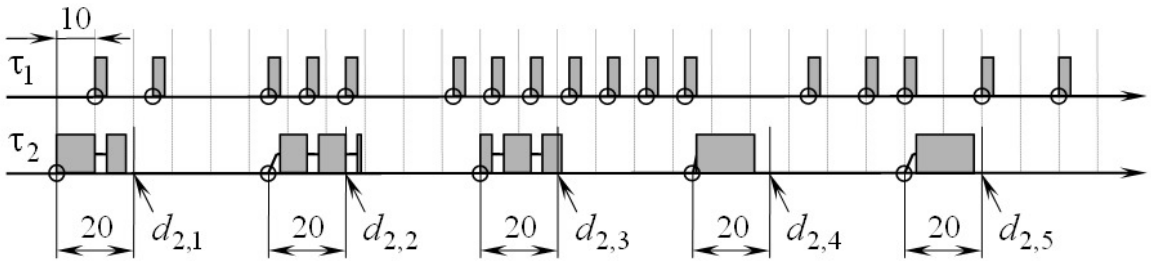

Figure 3.4: Some deadlines are violated for jobs with 60% WCET, i.e., when the hardware performance is increased roughly by 67%

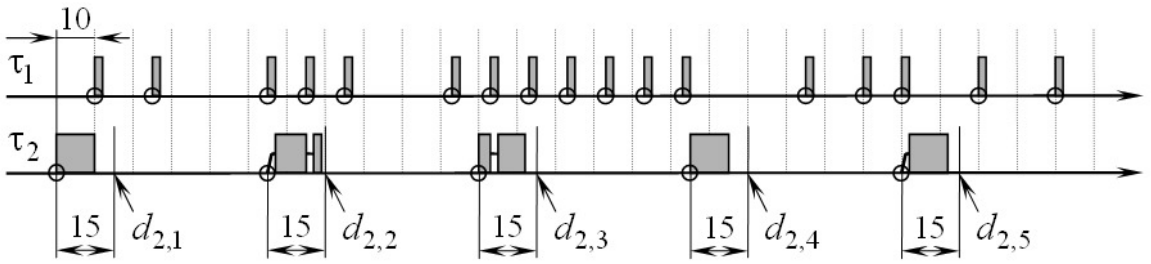

Figure 3.5: All deadlines are met for jobs with 40% WCET, i.e., when the hardware performance is increased roughly by 150%

increase the hardware performance. This can be easily checked by inspecting the execution of the task set shown in Figure 3.6 where intervals between consecutive jobs of τ_2 are represented. This intervals are in the range allowed by the original non-standard constraint described above.

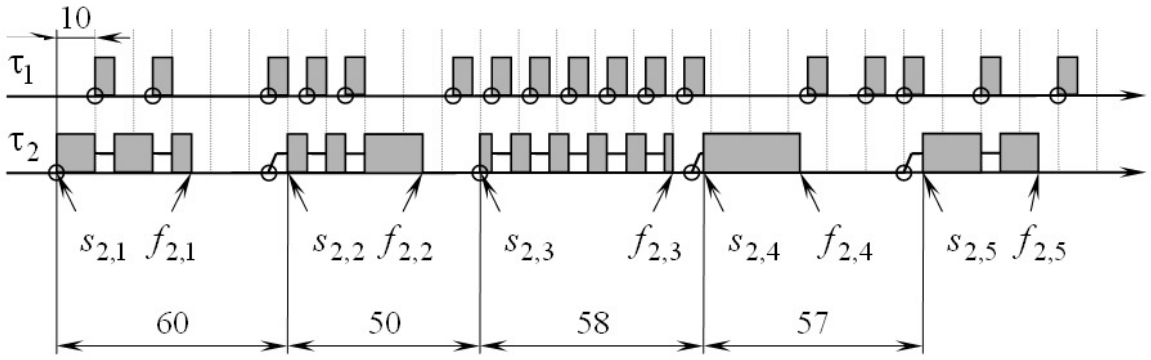

Figure 3.6: The original non-standard timing constraint is met without the increase in the hardware performance

This example gives us an intuition that the baseline attribute assignment can be improved by using the original non-standard timing constraints during off-line scheduling and in many case this improvement can be significant.

4 Conclusion

This paper shows that exploitation of the original non-standard timing constraints in the scheduling analysis and attribute assignment can lead to the increase of the success rate of scheduling for a given task set.

Thus we can describe the following problems to solve in order to exploit non-standard timing constraints in fixed priority scheduling.
(i) Propose a class of generalized timing constraints that includes both standard and non-standard timing constraints.
(ii) Develop the scheduling algorithms that include schedulability analysis and attribute assignment for tasks with these generalized timing constraints.
(iii) Evaluate the scheduling algorithms in comparison to the baseline approach described in this paper, see 2.4.1.

Bibliography

[1] N.C. Audsley, "Optimal Priority Assignment and Feasibility of Static Priority Tasks with Arbitrary Start Times," Technical Report YCS164, Department of Computer Science, University of York, 1991.

[2] G. Fohler, "Dynamic Timing Constraints — Relaxing Overconstraining Specifications of Real-Time Systems," *Proceedings of IEEE Real-Time Systems Symposium – Work-in-Progress Session*, 1997, pp. 27–30.

[3] J. Mäki-Turja and M. Nolin, "Efficient Response-Time Analysis for Tasks with Offsets," *Proceedings of IEEE Real-Time and Embedded Technology and Applications Symposium*, 2004, pp. 462–471.

[4] J. Mäki-Turja and M. Nolin, "Fast and Tight Response-Times for Tasks with Offsets," *Proceedings of Euromicro Conference on Real-Time Systems*, 2005, pp. 127–136.

[5] P. Martí, J.M. Fuertes, G. Fohler, and K. Ramamritham, "Improving Quality-of-Control Using Flexible Timing Constraints: Metric and Scheduling Issues," *Proceedings of IEEE Real-Time Systems Symposium*, 2002, pp. 91–100.

[6] P. Martí, J.M. Fuertes, K. Ramamritham, and G. Fohler, "Jitter Compensation for Real-Time Control Systems," *Proceedings of IEEE Real-Time Systems Symposium*, 2001, pp. 39–48.

[7] P. Martí, R. Villà, J.M. Fuertes, and G. Fohler, "Stability of On-Line Compensated Real-Time Scheduled Control Tasks," *IFAC Conference on New Technologies for Computer Control (NTCC01)*, Hong Kong, November, 2001.

[8] J.C. Palencia and M. González Harbour, "Schedulability Analysis for Tasks with Static and Dynamic Offsets," *Proceedings of IEEE Real-Time Systems Symposium*, 1998, pp. 26–37.

[9] K.W. Tindell, "An Extendible Approach for Analysing Fixed Priority Hard Real-Time Tasks," Technical Report YCS189, Department of Computer Science, University of York, 1992.

[10] K.W. Tindell, "Using Offset Information to Analyse Static Priority Pre-Emptively Scheduled Task Sets," Technical Report YCS182, Department of Computer Science, University of York, 1992.

[11] Sha, Lui and Abdelzaher, Tarek and Årzén, Karl-Erik and Cervin, Anton and Baker, Theodore and Burns, Alan and Buttazzo, Giorgio and Caccamo, Marco and Lehoczky, John and Mok, Aloysius K, "Real time scheduling theory: A historical perspective," Real-time systems, Vol. 28, 2004, pp. 101–155.

[12] Henriksson, Dan, and Anton Cervin. "Optimal on-line sampling period assignment for real-time control tasks based on plant state information." Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on. IEEE, 2005.

[13] Tindell, Ken W., Alan Burns, and Andy J. Wellings. "An extendible approach for analyzing fixed priority hard real-time tasks." Real-Time Systems 6.2 (1994): 133-151.

[14] Palencia, JosΓ© Carlos, and M. GonzΓΫlez Harbour. "Schedulability analysis for tasks with static and dynamic offsets." Real-Time Systems Symposium, 1998. Proceedings. The 19th IEEE. IEEE, 1998.

[15] Bini, Enrico, and Giorgio C. Buttazzo. "Schedulability analysis of periodic fixed priority systems." IEEE Transactions on Computers 53.11 (2004): 1462-1473.