Task Attribute Assignment of Fixed Priority Scheduled Tasks to Reenact Off-line Schedules

Radu Dobrin, Yusuf Özdemir, and Gerhard Fohler
Department of Computer Engineering
Mälardalen University, Sweden
{rdn99001@student, yor99002@student, gerhard.fohler@mdh.se}

Abstract

A number of industrial applications advocate the use of time triggered approaches for reasons of predictability, distribution, and particular constraints such as jitter or end-to-end deadlines. The rigid offline scheduling schemes used for time triggered systems, however, do not provide for flexibility. Fixed priority scheduling can provide more flexibility, but is limited with respect to predictability, as actual times of executions depend on run-time events.

In this paper, we present a method to combine offline schedule construction with fixed priority scheduling: by determining task attributes for the offline scheduled tasks, such that the original schedule is reconstructed if scheduled with FPS at run-time. It analyzes an offline schedule together with original task constraints to create sequences and windows of tasks. Priorities and offsets are set to ensure task orders in sequences and relation between windows. As FPS cannot reconstruct all schedules with periodic tasks, our algorithm can split tasks into several instances to achieve consistent task attributes. Lower priority tasks can be added for runtime use.

1 Introduction

Fixed priority scheduling (FPS) has been widely studied and used in a number of applications, mostly due by its simple run-time scheduling and resulting small overhead. Modifications to the basic scheme to handle semaphores [9], aperiodic tasks [10], static [11] and dynamic [7] offsets, and precedence constraints [6], have been presented. Consequently, FPS enables good flexibility for tasks with incompletely known attributes. Temporal analysis of FPS algorithms focuses on meeting deadlines, i.e., guarantees that all instances of tasks will finish before their deadlines. The actual times of executions of tasks, however, are generally not known and depend largely on run-time events, compromising predictability.

Off-line scheduling for time-triggered systems, on the other hand, provides strong predictability, as all times for task executions are determined and known in advance. In addition, complex constraints can be solved off-line, such as distribution, end-to-end deadlines, precedence, jitter, or instance separation. All this is enabled at the expense of losing run-time flexibility, as all actions have to be planned before.

In this paper, we present an algorithm to combine offline schedule construction with fixed priority run-time scheduling. The resulting systems have a time-triggered base that is complemented with even-triggered online scheduling. This allows us to combine benefits of offline scheduling, in particular a distributed system, complex, constrained tasks, and end-to-end deadlines, with online scheduling, which allows flexible task execution. A number of tasks are specified to execute predictable, while allowing flexibility for all others.

Our method works by transforming offline scheduled tasks with their original constraints into tasks with attributes suited for fixed priority scheduling, i.e., periods, deadlines, and offsets, which will reenact the original offline schedule at runtime. It divides the offline schedule and its tasks into windows and sequences, sets priorities to ensure execution orders within windows, and determines priorities and offsets to ensure orders and relations between windows. As FPS cannot reconstruct all schedules with periodic tasks, our algorithm can split tasks into several instances to achieve consistent task attributes. Tasks with lower priorities can be added for run-time scheduling.

Priority assignment for FPS tasks has been studied in, e.g., [1], [5], and [8] study the derivation of task attributes to meet a overall constraints, e.g., demanded by control performance. Instead of specific requirements, our algorithm takes an entire offline schedule and all task requirements to determine task attributes. A method to transform off-line schedules into earliest deadline first tasks has been presented in [4].

The rest of this paper is organized as follows: section 2 describes the problem and introduces items and terms...
used in the paper. Section 3 and 4 describe basic idea and algorithm, which is illustrated by an example in section 5. Summary and outlook in section 6 conclude the paper.

2 Problem Description

2.1 Offline Schedule

First, an off-line schedule is created for a set of tasks and constraints. While our method does not rely on a particular off-line scheduling algorithm, we have used the one described in [3] for our implementation and analysis. The schedule is usually created up to the least common multiple, LCM, of all task periods. LCM/T(T) instances of each task T with period T(T) will execute in the schedule.

The off-line scheduler resolves constraints such as distribution, end-to-end deadlines, precedence, etc, and creates scheduling tables for each node in the system, listing start- and finishing-times of all task executions. These scheduling tables are more fixed than required by the original constraints, so we can replace the exact start- and finishing-times of tasks with feasibility windows, taking the original constraints into account. A task receiving (sending) a message over the network, for example, has to start (finish) after (before) the scheduled transmission time, giving more leeway than the rigid scheduling table, defining release times (deadlines). Slot shifting [4] uses this method to transform off-line schedules into task for earliest deadline first scheduling.

Feasibility windows WF(T) of each instance j of a task i, provided by the task constraints expressed in the offline schedule.

The earliest start time, est(T), of an instance of a task i, is the time when T is starting its execution according to the offline schedule.

The scheduled finishing time, f(T), of an instance j of a task i, is the time when T is completing its execution according to the offline schedule.

The scheduled start time, start(T), of an instance j of a task i, is the time when T will start its execution according to the offline schedule. A sequence S(T) consists of instances of tasks T ordered increasing after their scheduled start times, such that Start(WF(T))=t and Start(WF(T))<t and f(T)>t, according to the sequence of execution in the offline schedule.

First S(T)=S(T) is first task instance in the sequence S(T)

Last S(T)=S(N) = last task instance in S(T)

S(k) = [S] S=T and k=instance, where j=task, j=instance,

start(WF(T))=t or start(WF(T))<t and f(T)>t, and start(S)=start(S)<start(S)<...start(S)=first(S) and S=last(S)}

2.2 Online Scheduling

At run-time, we want tasks to be scheduled according to fixed priority assignment. Our method assigns priorities, offsets, deadlines, periods.

We refer to an execution_window, Wexec(T), of an instance j of a task i, as the time interval in which T will execute if scheduled by FPS together with the other instances of the other tasks.

2.3 Problem Statement

Given a set of tasks T with earliest start times est(T), deadlines d(T), periods T(T), offline schedule, constraints represented by feasibility windows, we want to find fixed priorities, fixed offsets and deadlines for these tasks, P(T), O(T), d(T), such that the execution windows of each task Wexec(T) given by FPS will be contained within the respective feasibility window WF(T) and the sequence ordering kept.

In case of priority conflict, i.e., if assigning the same priorities to all instances of a task fail, we want to find priorities, offsets, and deadlines, such that each of these instances executes within its respective feasibility window for a small number of instances.

3 Overview – Basic Idea

3.1 Overview

Our method performs the following steps transforming the offline schedule into FPS tasks.

(1) Original constraints

(2) Offline Schedule

(3) Feasibility Windows

(4) Sequences of execution

Work before algorithm

(5) Transformation Algorithm

(6) Transformed Constraints

(7) FPS Tasks

(8) Execution Windows and Transformed Schedule

(1) Initially, tasks are given with their original constraints, and attributes, including worst case computation times, and periods.

(2) A standard offline scheduling-algorithm constructs offline-scheduling tables with (1) as input.
(3) Feasibility windows for each instance of each task are derived from scheduling tables and original task constraints.

(4) Sequences \( S(t_k) \subseteq WF(T_i^j) \) are now straightforward to derive from the feasibility windows.

(5) and (6) perform the actual attribute assignment creating FPS task (7). Whenever a priority conflict arises during the priority settings, we split a task \( T_i \) into instances for different priorities.

(8) Having the set of tasks with priorities, offsets, periods, deadlines, it is straightforward to schedule these using FPS.

3.2 Basic idea of attribute assignment algorithm

Our algorithm determines priorities by traversing the off-line schedule represented by the series of feasibility windows in increasing order of time (“left to right”). It sets priorities to reflect the positions of task instances in sequences, i.e., later position gives lower priority.

The algorithm attempts to keep the same priorities for all instances of a task. This may, however, lead to an inconsistent assignment. Then the algorithm splits the task into instances and assigns new attributes for each.

4 Algorithm

4.1 Assumptions

Our method is based on the following assumptions:

- Task deadlines are less than the task periods.
- The computation-time of a task has a known, upper bound.
- Tasks execute as soon as they are ready and have the highest priority.
- Tasks are independent and fully preemptive. All dependencies have been resolved in the off-line schedule.
- A higher numerical value for priority represents higher priority.

4.2 Attribute assignments

It suffices to set priorities according to the sequence of tasks of a feasibility window, if it does not overlap with any other feasibility window. In the case of overlaps, however, the order in more than one sequence has to be taken into account for the priority assignment.

Our algorithm traverses the schedule in the sequence of feasibility windows and performs two operations: set and check + split/update.

- Set is performed on each first instance \( j=1 \) of each task \( i \) by assigning a priority to the task \( T_i \), according to the sequence \( S(t_k) \), \( \omega = \text{Start}(WF(T_i^j)) \). For the rest of the instances of \( T_i \), we check if the priority assigned by ‘set’ will schedule \( T_i^j \) within it’s feasibility window in the same position in the sequence \( S(t_k) \) given by the offline schedule. It may happen, however, that the ordering of tasks may be different for some instances. These cases cannot be expressed directly with fixed priority assignment, leading to inconsistent priority assignment. Our algorithm detects this, and circumvents the problem by splitting a task into its instances by performing split and update on all the previous instances of \( T_i^j \) by updating offsets, periods and deadlines and by that, treating all instances of \( T_i \) as individual tasks.

So, the priorities of instances \( j \) of tasks \( T_i \) with \( \text{est}(T_i^j) = \text{start}(WF(T_i^j)) \) are based upon either:

- the priority of the first task instance in sequence \( S(t_k) \) in this window or
- the priority of another task assigned to this window which was set in a previous step.

Initially, no priorities are set: \( \forall i, \forall j, P(T_i^j) = \text{NULL} \), offsets are equals to start(WF(T_i^j)), \( \forall i, \forall j, O(T_i^j) = \text{start}(WF(T_i^j)) \), and deadlines are set to the ends of respective feasibility windows, \( \forall i, \forall j, D(T_i^j) = \text{end}(WF(T_i^j)) \). The priority of \( T_i^j \in S(t_k) = \{ T_1, T_2, \ldots, T_m \} \) must be set/checked to be in descending order: \( P(T_1) > P(T_2) > \cdots > P(T_m) \).

4.3 Task attribute assignment algorithm:

For each start of feasibility window \( t_k \), \( k \) number of feasibility windows,

\( \forall i, j \) such that \( \text{est}(T_i^j) = t_k \):

1) if \( j=1 \) // (the first instance of task \( T_i \))
   a) if \( \exists T_m^n \in S(t_k) \) and \( n \geq 1 \) (\( P(T_m^n) \neq \text{NULL} \))
      // \( P(T_m^n) \) is the priority of \( T_m \)
      // one instance in sequence has already pre-
      // set priority – set priorities in sequence
      // accordingly
      if \( T_m^n \) is scheduled to execute after \( T_i^j \) (according to the sequence \( S(t_k) \))
      \( \text{set} \ P(T_i^j) = P(T_m^n) + (x+1)-c \)
      // \( x = \) nr. of executing tasks belonging to \( WF(T_i^j) \) between \( T_i^j \) and \( T_m^n \) and \( c \) is a //constant.
      else
      // \( T_i^j \) is scheduled to execute before \( T_i^j \) // (according to the sequence \( S(t_k) \))
      \( \text{set} \ P(T_i^j) = P(T_m^n) - (x+1)-c \)
   b) else
      // \( \forall m, n \) such that \( T_m^n \in S(t_k) \), \( n \geq 1 \), (\( P(T_m^n) = \text{NULL} \) (only first instances in the // sequence \( S(t_k) \))
      \( \text{set} \ P(\text{first } S(t_k)) = P(S_i) = \text{default} = p \)
      // initial number priorities can be adjusted // after assignment, \( S_i = \text{First}(S(t_k)) \)
and
\[ P(T_i^j) = P(S_i) - x \cdot c. \]

//ix=number. of executing tasks (with
//higher priority) in/S(t_i) before T_i^j.
2) else
   // j > i, later instance
   // P(T_i^j) = P(T_i^j+1),...=P(T_i^k), priorities for
   //all instances have already been
   //set/checked in a previous step to the same
   //value

∀ m,n,n,i,n,j, such that T_m^j ∈ S(T_k) and P(T_m^j) ≠
NULL

// T_m^j has been assigned a priority in a
//previous step

Check priority of T_i^j
a) if T_m^j is scheduled to execute after (in time) T_i^j
   (according o the sequence S(t_i))
   if P(T_i^j) > P(T_m^j)
      OK!
   else // P(T_i^j) < P(T_m^j) => priority conflict
      Split/Update // 4.4

b) if T_m^j is scheduled to execute before
   T_i (according to the sequence S(t_i))
   if P(T_i^j) < P(T_m^j)
      OK!
   else // P(T_i^j) < P(T_m^j) => priority conflict!
      Split/Update

4.4 Priority assignment conflict – Instance splitting

Two instances, i and j, of two tasks T_m, T_n have a
priority-relation, P(T_m^i) > P(T_n^j), in an feasibility window
WF(T_m^i) or WF(T_n^j). Now, suppose that at a later point in
time the relation between these two priority-assignments is
contradicted: P(T_m^{i+k}) < P(T_n^{j+p}). Either T_m^{i+k} must
have a different priority than the previous instance T_m^i, or
T_n^{j+p} than T_n^j.

When a priority assignment conflict arises between
two instances T_m^i and T_n^j, we have to change the priority of
one of the instances, which causes the conflict, T_m or
T_n. One way to solve this problem is to split one of these
two task’s instances, T_m^i or T_n^j, k=1,2,... The result is that
the split instances will be treated as unique tasks with their
own attributes.

The selection of which task to split depends on the
task’s period and whether it is possible to split this task.
The first choice is to split the task with the largest period,
because that gives the least number of new tasks (least
number of instances). A task T_m can be split without
risking further conflicts, if the earliest start time of the
conflicting instance T_m^i is the start of the current
feasibility window WF(T_m^i):

If we split a task T_m whose instance T_m^i has become
feasible before the start of the current feasibility window,
a priority change of T_m^i could lead to another conflict in
one of it’s previous feasibility window where the task’s
priority has already been checked in a previous step. Further on, the successive instances of T_m will be treated
as first instances of new tasks.

When we split a task T_m, the attributes of all instances
of T_m^k, k=1,2,... will be updated, with:
• offset(T_m^k)=(k-1)*period(T_m)
• dl(T_m^k)=k*period(T_m)+dl(T_m)
• period(T_m^k) = LCM

5 Example

We illustrate the algorithm with an example. Let’s assume that we have the following task-set: A(15,2), B(15,1), C(15,5), D(10,3) where T(Period, computation time). The original offline schedule for the taskset is shown in figure 4.1

Fig 4.1

The feasibility windows for the instances of the tasks starts at t_1=0, t_2=10, t_3=15, t_4=20.

At time t=0, the instances whose earliest start times are equal to t are D, A, B, C. We set the default priority (in this case 100) on D and lower priorities of A, B and C according to the sequence S(t_1)={D, A, B, C}.

At time t=10, the second instance of D has its earliest start time equal to t_3. Since D has already been assigned the priority P(D)=100 at time t_1, we check if this priority will schedule the second instance of D into its off-line scheduled position according to S(t_2)={C, D}. In this case, since C’s already assigned priority P(C)=97 is lower than P(D)=100, we have a priority conflict. We have to choose which task to split. The first candidate is C because the number of invocations of C during LCM, 2, is lower then the number of invocations of D, 3. Since C’s earliest start time for this instance is, however, t_3<est(D), we have to split and update D, creating three tasks D1, D2, D3. The previous instance of D, D1, is updated with T(D1)=LCM, O(D1)=est(D)=0, dl(D)=end(WF(D1)) =10, P(D1)=100. D2. The instance creating the conflict,
D2, is treated as the first instance of a new task D2, with T(D2) = LCM, O(D2) = est(D2) = 10, dl(D2)= end(WF(D2))=20, and with a priority based on the priority of C, according to the sequence S(t2)={C, D2}, P(D2)=P(C)-1=96.

<table>
<thead>
<tr>
<th></th>
<th>A², B², C²</th>
<th>NONE</th>
<th>Check(A²)</th>
<th>OK</th>
<th>Check(B²)</th>
<th>OK</th>
<th>Check(C²)</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>D3</td>
<td>C²</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

At time t=20, we have the first instance of the new task D3 with its earliest start time equal to t, Since the task has no priority, we set P(D3) based on the priority of the C, according to the sequence S(t4)={C, D3}, P(D3)=P(C)-1=96.

The final set of tasks provided by our algorithm is shown in fig 4.2.

<table>
<thead>
<tr>
<th>task</th>
<th>C</th>
<th>T</th>
<th>O</th>
<th>dl</th>
<th>priority¹</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>15</td>
<td>0</td>
<td>15</td>
<td>99</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>15</td>
<td>0</td>
<td>15</td>
<td>98</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>15</td>
<td>0</td>
<td>15</td>
<td>97</td>
</tr>
<tr>
<td>D1</td>
<td>3</td>
<td>30</td>
<td>0</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>D2</td>
<td>3</td>
<td>30</td>
<td>10</td>
<td>20</td>
<td>96</td>
</tr>
<tr>
<td>D3</td>
<td>3</td>
<td>30</td>
<td>20</td>
<td>30</td>
<td>96</td>
</tr>
</tbody>
</table>

Fig 4.2

6 Summary and Outlook

In this paper, we presented a method to combine off-line schedule construction with fixed priority run-time scheduling. It uses off-line schedules to express complex constraints and predictability for selected tasks, while providing flexibility for the remaining tasks and newly added ones at run-time. At first, it analyses the off-line schedule together with the original task constraints and divides into windows and sequences. Then, it sets priorities and offsets to ensure task orders in sequences and relation between windows. The algorithm splits periodic tasks into several instances, as not all schedules can be expressed with FPS. Using fixed priority scheduling at run-time, the tasks with computed attributes will execute to "reenact" the original offline schedule. New, lower priority tasks can be added for run-time execution.

We have implemented the described methods and tested the results both via response time analysis and the original off-line schedule and original task constraints [2].

Our algorithm determines task attributes trying to keep priorities for all instances of periodic tasks the same. This will lead to inconsistent priority assignments for some schedules, e.g., those created with a earliest deadline first strategy. Our algorithm attempts to resolve the arising priority conflicts by splitting the task into several instances with different priorities. We are currently investigating this issue further to try to reduce the number of instances with different attributes created.

We assume here, that task dependencies have been resolved off-line. Future work will address the issue of task relations at run-time as well.

7 Bibliography


¹ Priorities can be adjusted to a desired range.