

Attribute Assignment for the Integration of Off-line and Fixed Priority Scheduling

Radu Dobrin and Gerhard Fohler
Department of Computer Engineering
Mälardalen University, Sweden
{radu.dobrin,gerhard.fohler}@mdh.se

Abstract

In this paper we present work to combine fixed priority scheduling with off-line schedule construction. It assumes a schedule has been constructed off-line for a set of tasks to meet their complex constraints. Our method takes the schedule and assigns the FPS attributes priority, offset, and period, to the tasks, such that their runtime FPS execution matches the off-line schedule. It does so by dividing the schedule into sequences and deriving priority inequalities, which are then resolved by integer linear programming. As FPS cannot reconstruct all schedules with periodic tasks, we have to split tasks into several instances to achieve consistent task attributes. Our algorithm constructs the minimum number of such artifact tasks.

1 Introduction

Off-line scheduling and fixed priority (FPS) scheduling are often considered as having incompatible paradigms, but complementing properties: off-line scheduling provides for predictability, distribution, and complex constraints such as jitter or end-to-end deadlines, but no runtime flexibility. FPS, on the other hand, provides this flexibility, but is limited to simple constraints, such as mutual exclusion.

In this paper, we present work to combine FPS with off-line schedule construction. It assumes a schedule has been constructed off-line for a set of tasks to meet their complex constraints. Our method takes the schedule and assigns the FPS attributes, priority, offset, and period, to the tasks, such that their runtime FPS execution matches the off-line schedule. It does so by dividing the schedule into sequences and deriving priority inequalities, which are then resolved by integer linear programming. Thus, our method combines FPS flexibility at runtime with the capability of off-line scheduling to resolve complex constrained tasks.

FPS cannot reconstruct all schedules with periodic tasks

with same priorities for all instances directly. The constraints expressed via the off-line schedule may require different task sequences for instances of the same tasks, as, e.g., by earliest deadline first, leading to inconsistent priority assignment, which can be expressed as a circle of inequalities.

Our algorithm detects such situations, and circumvents the problem by splitting a task into its instances, which are treated as tasks, and assigning different priorities to the newly generated “artifact” tasks, the formerly conflicting instances.

A key issue in resolving the priority conflicts is the number of artifact tasks created. Depending on where a priority conflict circle is “broken”, the number may vary, depending on the periods of the split tasks. Our algorithm constructs the minimum number of such artifact tasks.

Priority assignment for FPS tasks has been studied in, e.g., [1], [4], and [5] 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 off-line 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 [3]. A related paper [2] deals with priority assignment for offline schedules, but using only a constructive, heuristic approach, potentially creating large numbers of artifacts tasks.

2 Algorithm

Figure 1 gives an overview of the algorithm.

2.1 Overview

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

2) A standard off-line scheduling-algorithm constructs off-line scheduling tables with (1) as input.

Off-line scheduler

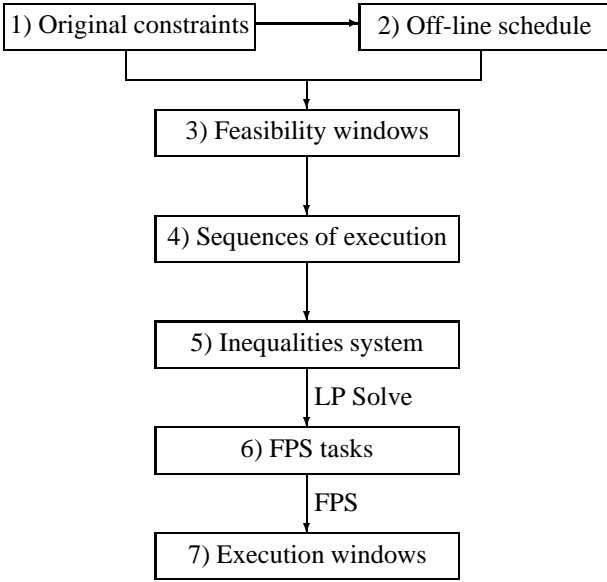


Figure 1. Algorithm overview.

3) Feasibility windows for each instance of each task are derived from the original task constraints and the off-line schedule.

4) Sequences are now straightforward to derive from the feasibility windows and the execution order expressed in the off-line schedule.

5) The analysis of each sequence provides a set of inequalities between priorities of instances of different tasks.

6) Using integer linear programming solves the system of inequalities and the result is the final set of tasks with fixed priorities.

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

Off-line 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. $LCM/T(T_i)$ instances of each task T_i with period $T(T_i)$ 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 origi-

nal constraints into account. A task receiving a message over the network, for example, has to start after the scheduled transmission time, giving more leeway than the rigid scheduling table, defining release times.

Feasibility windows ($WF(T_i^j)$) of each instance j of each task T_i , are derived from the off-line schedule and the original constraints transformed into earliest start times and deadlines.

The *earliest start time*, $est(T_i^j)$, of an instance j of a task T_i , is provided by the task constraints expressed in the off-line schedule. The *scheduled finishing time*, $f(T_i^j)$, of an instance j of a task T_i , is the time when T_i^j , is completing its execution according to the off-line schedule. The *scheduled start time*, $start(T_i^j)$, of an instance j of a task T_i , is the time when T_i^j is starting its execution according to the off-line schedule.

A **sequence** $S(t_k)$ consists of instances of tasks T_i^j ordered by increasing scheduled start times according to the off-line schedule. A sequence may contain instances such that $start(WF(T_i^j)) = t_k$, current instances of $WF(T_i^j)$, or instances of T_i^j from overlapping feasibility windows such that $est(T_m^n) < start(WF(T_i^j))$ and $f(T_i^j) > start(WF(T_i^j))$, interference instances of $WF(T_i^j)$. $first(S(t_k)) = S_1 =$ first task instance in the sequence $S(t_k)$, $last(S(t_k)) = S_N =$ last task instance in $S(t_k)$.

We refer to an *execution window*, $W_{exec}(T_i^j)$, of an instance j of a task i , as the time interval in which T_i^j will execute *at runtime* if scheduled by FPS. We want to find fixed priorities, offsets, and deadlines such that the execution window of each instance $W_{exec}(T_i^j)$ by FPS will be contained within the respective feasibility window $WF(T_i^j)$ of the off-line schedule and the sequence ordering kept.

2.2 Inequalities

Our algorithm derives relations (inequalities of priorities) among instances by traversing the off-line schedule represented by the series of feasibility windows in increasing order of time. It determines priority inequalities of instances according to the sequences $S(t_k)$ associated with feasibility windows, such that $P(S_1) > P(S_2) > \dots > P(S_N)$, where $S_1 = first(S(t_k))$ and $S_N = last(S(t_k))$. Note that the inequalities have to take into account relations between priorities of instances of the current feasibility window and possibly overlapping feasibility windows.

Our goal is to keep the same priority for all instances of each task. It may happen, however, that the ordering of tasks may be different for some instances. These cases cannot be expressed directly with fixed priorities. We solve this problem by splitting the task with the inconsistent priority assignment into instances and deriving new attributes for

each instance. At run-time, each of these instances is treated as a periodic task. We apply an integer linear programming solver to the derived system of priority inequalities to first identify which instances to split in order to minimize the total number of tasks created, and to derive priorities. The flexibility of the ILP solver allows for simple inclusion of other criteria via goal functions.

2.3 Minimizing the number of final tasks

The inequalities obtained from the execution order within the sequences, may form a circular chain of priority relations between instances of tasks, e.g.,

$$P(T_i^j) > P(T_m^n) > \dots > P(T_i^{j+k}) > \dots > P(T_m^{n+q})$$

We use a higher value to represent a higher priority.

In this case we cannot assign the same priority to both instances j and $(j+k)$ of T_i , nor to instances n and $(n+q)$ of T_m . We have to break the chain by splitting either T_i or T_m into instances and considering each one of these instances as individual tasks, which will result in a larger number of fixed priority tasks compared to the number of original off-line tasks. We formulate a goal function for an integer linear programming solver to identify the minimum amount of tasks with fixed priorities.

$$G = \#final_tasks = \#orig_tasks + \sum_{i=1}^N (|T_i| - 1) * b_i$$

where $\#final_tasks$ is the number of final tasks, $\#orig_tasks$ is the number of original tasks, $|T_i|$ = number of instances of T_i in LCM and b_i is a boolean variable associated to each task T_i , $b_i \in \{0, 1\}$. $b_i = 1$ means that T_i needs to be split into $|T_i|$ tasks. Thus, we determine the minimum amount of fixed priority tasks and information about the tasks that have to be split (b_i). If $b_i = 1$, we assign periods, offsets and deadlines to each instance j of T_i as follows:

- $offset(T_i^j) = (j - 1) * period(T_i)$,
- $dl(T_i^j) = j * period(T_i) + dl(T_i)$,
- $period(T_i^j) = LCM$.

Without splitting, offsets are set to the earliest start time and deadlines to the end of the feasibility window. Now, the computation of the priorities is straightforward for the solver.

3 Example

We illustrate the method with an example. Assume that we have the off-line schedule below with the taskset: A(5,1), B(10,3), C(20,8), (period, computation time).

A	B	B	B	C	A	C	C	C	C	B	B	B	A	C	A	C	C		
0				5					10				15					20	

The derivation of the inequalities is exemplified in the following table:

t_k	Current instances	Intf. inst.	$S(t_k)$	inequalities
0	A^1, B^1, C^1	None	A^1, B^1, C^1	$P(A^1) > P(B^1)$ $P(B^1) > P(C^1)$
5	A^2	C^1	A^2, C^1	$P(A^2) > P(C^1)$
10	A^3, B^2	C^1	B^2, A^3, C^1	$P(B^2) > P(A^3)$ $P(A^3) > P(C^1)$
15	A^4	C^1	A^4, C^1	$P(A^4) > P(C^1)$

At time $t_k=10$, we have the inequalities $P(B^2) > P(A^3)$ and $P(A^3) > P(C^1)$ added to the relations obtained at $t_1=0$ and $t_2=5$: $P(A^1) > P(B^1)$, $P(B^1) > P(C^1)$, and $P(A^2) > P(C^1)$. That gives a circular chain of priorities that must be solved: $P(A^1) > P(B^1), \dots, P(B^2) > P(A^3)$. In this case, we can either choose to split task A, or task B.

Splitting B will create two instances, one additional artefact task, splitting A will result in 4, three new ones. The integer linear programming solver with the goal function described in section 2.3 provides the solution:

- $b_A = b_C = 0$
- $b_B = 1$, meaning task B is to be split,
- $\#final_tasks = \#orig_tasks + \sum_{i=1}^N (|T_i| - 1) * b_i = 4$

Next, the final task attributes are derived:

task	C	T	offset	dl	priority
A	1	5	0	5	3
B1	3	20	0	10	4
B2	3	20	10	20	2
C	8	20	0	20	1

4 Summary, Discussion and Ongoing Efforts

In this paper, we presented work 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.

Our method analyzes the off-line schedule and determines inequalities between priorities of tasks in sequences. Using fixed priority scheduling at run-time, the tasks with computed attributes will execute according to the original off-line schedule. The algorithm splits periodic tasks into several instances, as not all schedules can be expressed with FPS. We use integer linear programming to resolve

the inequality systems represented by the off-line scheduled tasks. The goal is to create the minimum number of artifact tasks.

To this point, we have concentrated on reconstructing the off-line schedule. We are investigating the inclusion of additional FPS tasks at runtime, which demands inclusion during priority assignment. Furthermore, we assume all task dependencies have been resolved off-line. Future work will address the issue of task relations at run-time as well.

We take task periods and the entire LCM as input. We are investigating using an ILP representation to derive unknown task periods as well, and to identify minimum repeating patterns to minimize considered schedule length and number of tasks.

The version presented here may split tasks into instances, but leaves instances intact. In certain cases, breaking up instances into junks might result in smaller total number of artifact tasks created.

5 Acknowledgements

The authors wish to express their gratitude to Peter Puschner for his assistance with ILP, and Yusuf Ozdemir, Pau Marti, Damir Isovich and Tomas Lennvall for useful discussions and comments on the paper.

References

- [1] N. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical report, Department of Computer Science, University of York, 1991.
- [2] R. Dobrin, Y. Ozdemir, and G. Fohler. Task attribute assignment of fixed priority scheduled tasks to reenact off-line schedules. In *Conference on Real-Time Computing Systems and Applications, Korea*, December 2000.
- [3] G. Fohler. *Flexibility in Statically Scheduled Hard Real-Time Systems*. PhD thesis, Technische Universität Wien, Austria, Apr. 1994.
- [4] R. Gerber, S. Hong, and M. Saksena. Guaranteeing real-time requirements with resource-based calibration of periodic processes. *IEEE Transactions on Software Engineering*.
- [5] D. Seto and L. Lehoczky. Task period selection and schedulability in real-time systems. In *Proceedings of Real-Time Systems Symposium*, 1998.