Online Handling of Firm Aperiodic Tasks in Time Triggered Systems

Damir Isović and Gerhard Fohler
Department of Computer Engineering, Mälardalen University, Sweden
{dic,gfr}@mdh.se

Abstract

A number of industrial applications advocate the use of time triggered approaches for reasons of predictability, cost, product reuse, and maintenance. The rigid offline scheduling schemes used for time triggered systems, however, do not provide for flexibility. At runtime, aperiodic tasks can only be included into the unused resources of the offline schedule, supporting neither guarantees nor fast response times.

In this paper we present an algorithm for flexible hard aperiodic task handling in offline scheduled systems: it provides an O(N) acceptance test to determine if a set of aperiodics can be feasibly included into the offline scheduled tasks, and does not require runtime handling of resource reservation for guaranteed tasks. Thus, it supports flexible schemes for rejection and removal of aperiodic tasks, overload handling, and simple reclaiming of resources. As a result, our algorithm provides for a combination of offline scheduling and online hard aperiodic task handling.

1 Introduction

Time triggered real-time systems have been shown to be appropriate for a variety of critical applications: they provide verifiable timing behavior and allow for distribution, complex application structures, and general constraints, such as precedence or end-to-end deadlines. Their benefits are, however, limited for applications with not completely known characteristics, such as arrival times. Including such non periodic tasks in the rigid offline schedules, as used for time triggered systems, can be resource inefficient at best for sporadic tasks, and impossible for aperiodics, i.e., without knowledge of arrival times.

Real-world applications demand flexible handling of aperiodic tasks: efficient acceptance tests, resource reservation if a task can be guaranteed, but also specific rejection strategies for the negative case. Often, aperiodics are guaranteed on a first-come-first-serve basis, i.e., already guaranteed tasks will be executed, and only newly arrived ones rejected, implying that earlier arriving tasks are more important. Instead, the selection of which tasks to reject or remove should be left to the designer, allowing values and importance assigned to tasks, not on their arrival times.

In this paper, we present methods for the flexible online handling of firm aperiodic tasks based on offline constructed schedules for time triggered systems. It is based on slot shifting, [4], a method to combine offline and online scheduling by utilizing unused resources. We provide an EDF based acceptance test of $O(N)$ to determine the feasible inclusion of aperiodic tasks into the offline scheduled tasks. The algorithm avoids explicit runtime handling of resource reservations for the guaranteed tasks and their impact on the offline scheduled tasks. Therefore, flexible schemes for rejection and removal of tasks and aperiodic overload handling can be applied. The resources of tasks completed earlier can be reclaimed for other aperiodic tasks without further provisions or explicitly freeing them.

Aperiodic task handling has been studied extensively for many scheduling schemes. Server based algorithms, for example, have been presented for earliest deadline, e.g., [10], and fixed priority systems, e.g., [9]. Example algorithms for the selection of tasks to reject in overload situations have been discussed in [3], [7],[2], [1]. These algorithms assume control over all tasks in the system and do not take into account the impact of offline scheduled tasks. The algorithm presented in [4] for guarantees of single firm aperiodic tasks on offline schedules does not provide for removal of guaranteed tasks.

2 Offline Complexity Reduction

The rigid offline scheduling schemes used for time triggered systems do not provide for flexibility. Including non periodic tasks in the offline schedule can be impossible for aperiodics, i.e., without any knowledge or restriction on arrival times. At runtime, aperiodic tasks can only be included into the unused resources of the offline schedule, supporting neither guarantees nor short response times.

In this section, we briefly describe the slot shifting method [4] which we use as a basis to combine offline and online scheduling. It provides for the efficient handling of aperiodic tasks on top of a distributed schedule with general task constraints. Slot shifting extracts information about unused resources and leeway in an offline schedule and uses this information to add tasks feasibly, i.e., without violating

requirements on the already scheduled tasks. A detailed description can be found in [4].

First, a standard offline scheduler, e.g., [8], or [5] creates scheduling tables for the periodic tasks. It allocates tasks to nodes and resolves precedence constraints by ordering task executions.

The scheduling tables list fixed start- and end times of task executions, eliminating all flexibility. The only assignments fixed by the specification of the tasks' feasibility, however, are the initiating and concluding tasks in the precedence graph, and, as we assume message transmission times to be fixed here, tasks sending or receiving inter-node messages. These are the only fixed start-times and deadlines, all others are calculated recursively during offline preparations. The execution of all other tasks may vary within the precedence order, i.e., they can be shifted.

The deadlines of tasks are then sorted for each node and the schedule is divided into a set of *disjoint execution intervals* for each node. Spare capacities are defined for these intervals.

Each deadline calculated for a task defines the end of an interval I_i, $end(I_i)$. Several tasks with the same deadline constitute one interval. The spare capacities of an interval I_i are calculated as given in formula 1:

$$sc(I_i) \;=\; |I_i| - \sum_{T \in I_i} MAXT(T) + min(sc(I_{i+1}), 0) \quad (1)$$

The length of I_i minus the sum of the activities assigned to it is the amount of idle times in that interval. These have to be decreased by the amount "lent" to subsequent intervals: Tasks may execute in intervals prior to the one they are assigned to. Then they "borrow" spare capacity from the "earlier" interval.

After determination of intervals and spare capacities, the offline preparations are completed and the amount and location of unused resources is available for online use.

Online scheduling is performed locally for each node. If the spare capacities of the current interval $sc(I_c) > 0$, EDF is applied on the set of ready tasks. $sc(I_c) = 0$ indicates that a guaranteed task has to be executed or else a deadline violation in the task set will occur. Soft aperiodic tasks, i.e., without deadline, can be executed immediately if $sc(I_c) > 0$. After each scheduling decision, the spare capacities of the affected intervals are updated.

3 Motivation and Approach

Guaranteeing and handling of firm aperiodic tasks involves three steps:

Acceptance test: Upon arrival of a firm aperiodic task, a test determines whether there are enough resources available to include it feasibly with respect to static and previously guaranteed aperiodic tasks.

Reservation of resources: If the task can be accepted, it is guaranteed by providing a mechanism which ensures that the resources it requires will be available for its execution. This can be achieved, e.g., by removing these resources from the available ones, or by ensuring that subsequent guarantees will not remove them. Note that acceptance test and guarantee can be separated.

Rejection strategy: A failed acceptance test indicates an overload situation. Consequently, a rejection strategy is required, which determines which task or tasks – out of all guaranteed or newly arrived tasks – to reject or abort.

3.1 Shortcommings of Previous Version

The original version of slot shifting provides an online guarantee algorithm of $O(N)$ as well: upon arrival of a firm aperiodic task A, the spare capacities up to its deadline are summed up and compared to the execution time demand. If A is accepted, the spare capacities and intervals are recalculated taking into account that resources needed for A are not available for other tasks. If the deadline of task A does not overlap with one of offline calculated intervals, then the interval that contains $dl(A)$ needs to be split. However, we want to avoid the creation of new intervals, in order to keep the online mechanism as simple as possible. While this algorithm guarantees single aperiodics, it has limited flexibility: only the task currently tested is possibly rejected; once guaranteed, aperiodics will execute. Changes in the set of guaranteed tasks require costly deletion of intervals, recalculation of spare capacities, and new guarantees.

3.2 Basic Idea

The new method presented here separates acceptance and guarantee. It eliminates the online modificiation of intervals and spare capacities and thus allows rejection strategies over the entire aperiodic task set. The basic idea behind the method is based on standard earliest deadline first guarantee, but sets it to work on top of the offline schedule: EDF is based on having full availability of the CPU; we have to consider interference from offline scheduled tasks and pertain their feasibility.

4 Algorithm Description

Let $\mathcal{G}_{t_1}$ be a set of guaranteed firm aperiodic tasks at time t_1, i.e., $\mathcal{G}_{t_1} = \{F_i \mid c_{t_1}(F_i) \geq 0 \wedge t_1 < dl(F_i) \leq dl(F_{i+1})\}$, where $c_{t_1}(F_i)$ denotes the remaining execution time of task F_i at time t_1, and $dl(F_i)$ the absolute deadline. Tasks in $\mathcal{G}_{t_1}$ are ordered by increasing deadlines, each later than t.

Now assume a new firm aperiodic task A arrives at time t_2. From time t_1 to t_2, some tasks in $\mathcal{G}_{t_1}$ could have executed up to t_2, which is reflected as follows:

- $\{F_1, ..., F_{k-1}\}$ – tasks completed by time t_2.

- F_k – task currently ready to run, according to EDF. It may have executed partially , so we need only to consider its remaining execution time, $c_{t_2}(F_k) \leq c(F_k)$.

- $\{F_{k+1}, ..., F_n\}$ – not yet started tasks that need to execute fully, $c_{t_2}(F_k) = c(F_k)$.

So, when guaranteeing A at time t_2, we need not consider already completed tasks, but only $\mathcal{G}_{t_2} \subset \mathcal{G}_{t_1}$, $\mathcal{G}_{t_2} = \{F_k, F_{k+1}, ...F_n\}$. Task A is accepted if the set $\mathcal{G}' = \mathcal{G}_{t_2} \cup A$ is feasible, considering the offline tasks.

4.1 Acceptance Test for a Set of Aperiodic Tasks

Offline scheduled tasks are guaranteed to complete before their deadlines. Aperiodic tasks use unused resources in the offline schedule. The amount and location of available resources are represented as intervals and spare capacities.

Let $\mathcal{F} = \{F_1, F_2, ..., F_n\}$ be a set of firm aperiodic tasks that need to be scheduled together with the offline tasks. We accept it if each task in $\mathcal{F}$ is guaranteed to complete before its deadline, i.e., the following must hold:

$$\forall i, 1 \leq i \leq n: \; c(F_i) \leq \begin{cases} \text{sc}[t, dl(F_1)] & , i = 1 \\ \text{sc}[ft(F_{i-1}), dl(F_i)] & , i > 1 \end{cases}$$

where t is current time and notation $sc[t_1, t_2]$ means the spare capacity from time t_1 to time t_2. Otherwise, we need to reject some task(s).

Note that the spare capacities are not distributed in a uniform way throughout the schedule. Rather, as described in 2, the schedule is divided into intervals, each with an individual value of spare capacity. Consequently, the amount of spare capacity in a window depends on the position of that window in the schedule.

The finishing time of a firm aperiodic task F_i is calculated with respect to the finishing time of the previous task, F_{i-1}. Without any offline tasks, it is calculated the same as in EDF algorithm i.e., $ft(F_i) = ft(F_{i-1}) + c(F_i)$. Since we guarantee firm aperiodic tasks on the top of an offline schedule, we need to consider the feasibility of offline tasks. This extends the formula above with a new term that reflects the amount of resources reserved for offline tasks in a certain time interval, $R[t_1, t_2]$:

$$ft(F_i) = c_t(F_i) + \begin{cases} t + \text{R}[t, ft(F_1)] & , i = 1 \\ ft(F_{i-1}) + \text{R}[ft(F_{i-1}), ft(F_i)] & , i > 1 \end{cases}$$

We can access $R[t_1, t_2]$ via spare capacities and intervals at runtime as $R[t_1, t_2] = (t_2 - t_1) - sc[t_1, t_2]$. Since $ft(F_i)$ appears on both sides of the equation, a simple solution is not possible. Rather, we present an algorithm for computation of finishing times of firm aperiodic tasks with complexity of $O(N)$, which is further discussed in next subsection.

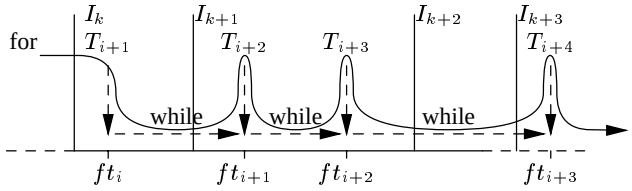
4.2 Pseudo Code

Here is the pseudo code for the acceptance test and algorithm for finishing time calculation:

```
ft = getFinishingTime(max(ft(A_{i-1}), t), c(A_i));
/* chech if accepting A_i causes any of previously
guaranteed firm aperiodic tasks to miss its deadline */
if(ft ≤ dl(A_i))
    for(j = i + 1; j < n; j++)
        ft = getFinishingTime(ft, c_rem(G_j));
        if(ft > dl(G_j)) ⇒ not feasible!
    insert(A, G);
else reject A;

getFinishingTime(ft_p, c_rem)
  /* determine ft by "filling up"
  free slots until the c_rem is exhausted. */
  sc_rem = start(I_c) + sc(I_c) − ft_p;
  while (c_rem > sc_rem)
      if(sc_rem (I_c) > 0)
          c_rem = c_rem − sc_rem;
      c++;
      ft_p = start(I_c)
      sc_rem = sc(I_c);
  return (ft_p + c_rem);
```

The complexity of algorithm above is $O(N)$, because we go through all tasks only once, and calculate their finishing times on the way, as depicted bellow.



The *for-loop* picks a task and start the *while-loop*, which calculates its finishing time by going through the intervals. We do not have any nested loops, and we always continue forward.

4.3 Resource Reservation

The method presented here reserves resources implicitly, by only accepting a new task if it can be guaranteed *together* with all previously guaranteed ones. Consequently, removal of guaranteed tasks and changes in the set of tasks can be handled efficiently.

4.4 Rejection Strategies and Overload Handling

Our method allows for easy changes in the set of guaranteed tasks and thus supports rejection strategies and overload handling mechanisms. It allows a new set of candidates to be

submitted to the acceptance test and does not require modifications to the reserved resources for guaranteed tasks. We are currently investigating the application of overload handling schemes, such as presented in [3], [1].

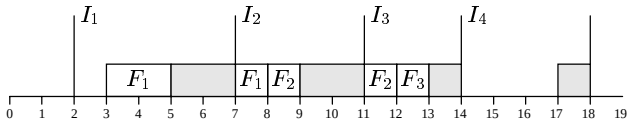
4.5 Resource Reclaiming

Should aperiodic tasks use less resources than expressed in worst case parameters, our method directly reclaims these without recalculation of available resources. Rather, the next time the acceptance test is performed, the fact that a task has an earlier finishing time is considered in the calculations.

5 Example

Let $\mathcal{G}_3 = \{F_1(3,10), F_2(2,18), F_3(1,19)\}$ be the set of previously guaranteed but not completed firm aperiodic tasks at current time $t = 3$, where the first parameter is the remaining execution time, the second absolute deadline. Tasks in $\mathcal{G}_3$ are ordered by increasing deadlines and they will execute in the first available slots, in EDF order.

At time $t = 3$ we have the execution scenario of both offline scheduled tasks (the shaded boxes) and guaranteed aperiodic tasks from $\mathcal{G}_3$ as follows:

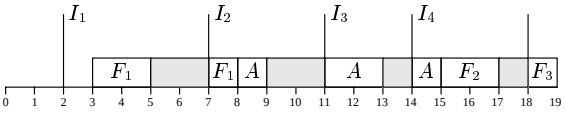


Now assume a firm aperiodic task $A(4,16)$ arrives at $t = 3$. We perform the online guarantee algorithm:

1. Task F_1 has earlier deadline than A, so F_1's position in the set $\mathcal{G}_3$ remains unchanged, i.e., before A.

2. Task F_2 has a deadline after the deadline of A, which means that the A should execute before F_2. We check if A can complete before its deadline: $ft_A = getFT(ft_{F_1}, c(A)) = 15 < 16 \Rightarrow yes!$

3. Now we must check if accepting task A will cause any of other guaranteed firm aperiodic tasks (F_2, F_3) to miss their deadlines. We calculate their finishing times: $ft_{F_2} = 17 < 18$, $ft_{F_3} = 19 \leq 19$. Both F_2 and F_3 can complete before their deadlines, which means that the new task A can be guaranteed and therefore inserted in the set of guaranteed firm aperiodic tasks $\mathcal{G}_3$:



6 Summary and Outlook

In this paper we presented an algorithm for the flexible handling of firm aperiodic tasks in offline scheduled systems. It is based on slot shifting, a method to combine offline and online scheduling methods.

First, a standard offline scheduler constructs a schedule, resolving complex task constraints such as precedence, distribution, and end-to-end deadlines. This is then analyzed for unused resources and leeway in task executions. The run-time scheduler uses this information to handle aperiodic tasks, shifting other task executions ("slots") to reduce response times without affecting feasibility. We provided an $O(N)$ acceptance test for a set of aperiodic tasks on the offline schedule and guarantee tasks without explicit reservation of resources. Our method supports flexible, value based selections of tasks to reject or remove in overload situations, and simple resource reclaiming.

While the current algorithm enables the rejection and removal of tasks, it does not address the issue of selection. We are investigating into providing a number of overload handling strategies, e.g., [3], [1].

In a previous paper [6], we presented an offline test for sporadic tasks based on worst case arrival assumptions. It cannot utilize less frequent arrivals for firm aperiodic tasks since the runtime overheads to reflect the continuous changes in resource availability are prohibitively high. We are looking into applying the algorithm presented here to handle sporadic tasks at runtime.

References

[1] S. A. Aldarmi and A. Burns. Dynamic value-density for scheduling real-time systems. In *Proceedings 11th Euromicro Conference on Real-Time Systems*, Dec 1999.

[2] S. Baruah, G. Koren, D. Mao, and B. Mishra. On the competitiveness of on-line real-time task scheduling. *Real- Time Systems*, 2(4), June 1992.

[3] G. Buttazzo and J. Stankovic. *Adding Robustness in Dynamic Preemptive Scheduling*. Kluwer Academic Publishers, 1995.

[4] G. Fohler. Joint scheduling of distributed complex periodic and hard aperiodic tasks in statically scheduled systems. In *Proc. 16th Real-time Systems Symposium*, Pisa, Italy, 1995.

[5] G. Fohler and C. Koza. Heuristic scheduling for distributed real-time systems. Technical Report 6/98, Institut für Technische Informatik, Technische Universität Wien, April 1989.

[6] D. Isovic and G. Fohler. Handling sporadic tasks in off-line scheduled distributed hard real-time systems. *Proc. of 11th EUROMICRO conf. on RT systems, York, UK*, June 1999.

[7] G. Koren and D. Shasha. Skip-over: Algorithms and complexity for overloaded systems that allow skips. In *Proceedings of the Real-Time Systems Symposium*, Dec. 1992.

[8] K. Ramamritham. Allocation and scheduling of complex periodic tasks. In *10th Int. Conf. on Distributed Computing Systems*, 1990.

[9] B. Sprunt, L. Sha, and J. Lehoczky. Aperiodic task scheduling for hard-real-time systems. *Real- Time Systems*, 1, July 1989.

[10] M. Spuri and G. Buttazzo. Efficient aperiodic service under earliest deadline scheduling. In *Proceedings of the Real-Time Systems Symposium*, Dec. 1994.