Proceedings of the 8th IEEE Int. Conf. on Emerging Technologies and Factory Automation Nice, France, October 2001.

Improved Handling of Soft Aperiodic Tasks in Offline Scheduled Real-Time Systems using Total Bandwidth Server

Gerhard Fohler, Tomas Lennvall
Mälardalen University
Västeras, Sweden
{gfr, tlv}@mdh.se

Giorgio Buttazzo
Univerisity of Pavia
Pavia, Italy
giorgio@sssup.it

Abstract - **Real-world industrial applications impose complex constraints, such as distribution, end-to-end deadlines, and jitter control on real-time systems. Most scheduling algorithms concentrate on single or limited combinations of constraints and requirements only. Offline scheduling resolves complex constraints, but provides only very limited flexibility. Online scheduling on the other hand, supports flexibility, resource reclaiming, and overload handling, but handling constraints such as distribution or end-to-end deadline can be costly, if not intractable.**

In this paper, we propose a method to efficiently handle soft real-time tasks in offline scheduled real-time systems using total bandwidth server. In a first step, the offline scheduler resolves complex constraints, reduces their complexity, and provides for guaranteed available bandwidth. The constructed schedule is translated into independent tasks on single nodes with starttimes and deadline constraints only. These are then executed using earliest deadline first, total bandwidth server scheduling at runtime.

I Introduction

Real-world industrial applications impose complex constraints on real-time systems and their scheduling algorithms: Even simple control problems for industrial plants or automotive scenarios demand distribution, end-to-end deadlines, jitter control, periodic and non periodic tasks, as well as efficiency, cost effectiveness and more. In addition to these basic temporal constraints, a system has to fulfill complex application demands which cannot be expressed as generally. Control applications may require constraints on individual instance, rather than periods. Reliability demands can enforce allocation and separation patterns, or engineering practice may require relations between system activities, all of which cannot be expressed directly with basic constraints.

Furthermore, the type and number of constraints rarely remains fixed during the development of a system or product line. Rather, new constraints, often beyond periods and deadlines, are added during system construction or its maintenance.

Most scheduling algorithms presented concentrate on single or limited combinations of constraints and requirements only. Two main lines of algorithms, following the paradigms of performing scheduling *offline*, i.e., before the system runtime, or *online*, excel at general constraints or flexibility, resp. Offline scheduling is capable of constructing schedules for distributed applications with complex constraints, such as precedence, jitter, and end-to-end deadlines. As only a table lookup is necessary to execute the schedule, runtime dispatching is very simple. The prerequisite knowledge about all system activities and events may be hard or impossible to obtain. The lack of flexibility prevents handling not completely specified events.

Online scheduling overcomes this shortcoming and provides flexibility for partially or non specified activities. Feasibility tests determine whether a given task set can be feasibly scheduled according to the rules of the particular algorithm applied. Online scheduling allows to efficiently reclaim any spare time coming from early completions and allows to handle overload situations according to actual workload conditions.

These feasibility tests typically apply to a fixed set of constraints; changes in the set of constraints often require development of theory for tests. Handling constraints such as distribution or end-to-end deadline can be costly, if not intractable in the general case.

This disparity of capabilities of scheduling paradigms imposes the choice of choosing the benefits of either algorithm at the expense of the other's, between general constraints or runtime flexibility. Rather, the application under consideration should dictate constraints and properties, with the choice of algorithm being only a secondary one.

In this paper, we propose a method to efficiently handle soft real-time tasks in offline scheduled real-time systems using total bandwidth server. It uses offline scheduling to handle complex constraints and reduce their complexity by transforming a constructed feasible schedule into independent tasks on single nodes with start times and deadline constraints only. These are suited for flexible earliest deadline first scheduling methods at runtime. Furthermore, the offline scheduler guarantees that the resulting task set guarantees the availability of a minimum bandwidth for use at runtime, extending the range of applicable methods and constraints. Via transformed task set and bandwidth, our method provides an interface to combine offline and online scheduling algorithms.

The concrete offline scheduler used for this work [6] allows for preemptive tasks with precedence constraints, end-to-end deadlines, distribution, network scheduling and jitter

control. The transformation technique we present can be applied to a variety of other offline scheduling algorithms with similar constraints, e.g., [10]. The inclusion of additional constraints into an offline scheduler is typically straightforward, e.g., by including the constraint in a feasibility test applied during schedule construction.

Online scheduling is used to efficiently handle those activities that cannot be completely characterized offline in terms of worst-case behavior, and hence cannot receive *a priori* guarantee. Examples of these activities include soft aperiodic tasks (e.g., multimedia tasks) whose computation time or interarrival times can have significant variation from instance to instance. Moreover, online scheduling is used to reclaim any spare time coming from early completion. A bandwidth reservation technique [1] is used to isolate the temporal behavior of the two schedules and prevent the event-driven tasks to corrupt the off-line plan.

The MARS system [8] is an example of a system with entire offline planning of all activities. On the other side of the spectrum, SPRING [14] is using planning and global task migration [16] for handling a variety of constraints online. Its planning efforts are expensive; a dedicated scheduling chip is suggested. In our approach, the online scheduling is very simple as we only compute new deadlines.

The use of free resources in offline constructed schedules for aperiodic tasks has been discussed in [11]. The resulting flexibility is limited since aperiodic tasks are inserted into the idle times of the schedule only. Slot shifting [7] analysis offline schedules for unused resources and leeway, which is represented as execution intervals and spare capacities. This information is used at runtime to shift task executions, accommodate dynamic tasks, and to perform online guarantee tests. It provides increased flexibility, but focuses on hard and firm tasks only.

From the online side, the integration of different scheduling paradigms in the same system requires a resource reservation mechanism to isolate the temporal behavior of each schedule. In [9], Mercer, Savage, and Tokuda propose a scheme based on processor capacity reserves, where a fraction of the CPU bandwidth is reserved to each task. This approach removes the need of knowing the worst-case computation time (WCET) of each task because it fixes the maximum time that each task can execute in each period. Since the periodic scheduler is based on the Rate Monotonic algorithm, the classical schedulability analysis can be applied to guarantee hard tasks, if any present.

In [5], Liu and Deng describe a two-level scheduling hierarchy which allows hard real-time, soft real-time, and non real-time applications to coexist in the same system. According to this approach, each application is handled by a dedicated server, which can be a Constant Utilization Server [4] for tasks that do not use nonpreemptable sections or global resources, and a Total Bandwidth Server [13, 12] for the other tasks. At the lowest level, all jobs coming from the different applications are handled by the EDF scheduling algorithm. Although this solution can isolate the effects of overloads at the application level, the method requires the knowledge of the WCET even for soft and non real-time tasks.

The use of information about amount and distribution of unused resources for non periodic activities is similar to the basic idea of slack stealing [15], [3] which applies to fixed priority scheduling. Our method applies this basic idea in the context of offline and EDF scheduling. Chetto and Chetto [2] presented a method to analyze idle times of periodic tasks based on EDF. Our scheme analyzes offline schedules, which can be more general than strictly periodic tasks, e.g., for control applications.

The rest of this paper is organized as follows: First, we define terminology in section II. The techniques for integration of offline and online are presented in section III. An example in section IV illustrates the methods. We conclude the paper with section VI.

II Terminology and assumptions

We consider a system consisting of three types of tasks: hard, soft, and non real-time tasks. Any task τ_i consists of a sequence of jobs $J_{i,j}$, where $r_{i,j}$ denotes the arrival time (or request time) of the j^{th} job of task τ_i.

A hard real-time task is characterized by two additional parameters, (C_i, T_i), where C_i is the WCET of each job and T_i is the minimum interarrival time between successive jobs, so that $r_{i,j+1} \geq r_{i,j} + T_i$. The system must provide an a priori guarantee that all jobs of a hard task must complete before a given deadline $d_{i,j}$. In our model, the absolute deadline of each hard job $J_{i,j}$ is implicitly set at the value $d_{i,j} = r_{i,j} + T_i$.

A soft real-time task is also characterized by the parameters (C_i, T_i), however the timing constraints are more relaxed. In particular, for a soft task, C_i represents the *mean* execution time of each job, whereas T_i represents the *desired* activation period between successive jobs. For each soft job $J_{i,j}$, a soft deadline is set at time $d_{i,j} = r_{i,j} + T_i$. Since mean values are used for the computation time and minimum interarrival times are not known, soft tasks cannot be guaranteed a priori. In multimedia applications, soft deadline misses may decrease the QoS, but do not cause critical system faults.

Tasks can be synchronized via *precedence constraints*, forming execution chains with end-to-end constraints. Precedence constraints and tasks, can be viewed as a directed acyclic graph. Tasks are represented as nodes, precedence constraints as edges. Tasks that have no predecessors are called *entry* tasks, tasks without successors *exit* tasks. The *period* of a precedence graph PG is the time interval separating two successive instances of PG in the schedule. *Deadline intervals* are defined for the maximum execution of each individual precedence graph. Entry tasks of instance i of a precedence graph PG become ready for scheduling at time $i \times period(PG)$. Exit tasks of instance i of a precedence graph PG have to be completed by time $i \times period(PG) + deadline(PG)$.

We consider a *distributed* system, i.e., one that consists of several *processing nodes* and *communication nodes* An *offline schedule* is a sequence of slots, i.e., some time granules, $slot_i$ $i = 0, \ldots,$ N-1, N stands for the number of slots in the schedule. For online schedules, N is typically equal to the least common multiple (lcm) of all involved periods.

Communication nodes, i.e., the communication medium, are slotted and pre scheduled as well. Protocols with bounded transmission times, e.g., TDMA or token ring are applicable.

III Integration

A Rationale

The rationale of our method to provide for complex application constraints and efficient runtime flexibility is to concentrate all mechanisms to handle complex constraints in the *offline* phase, where they are transformed into *simple constraints* suitable for earliest deadline first scheduling, which is then used for *online* execution. The offline determined simple constraints serve as "interface" between offline preparations and online scheduling. Specifically, we use the offline scheduler presented in [6][1], starttime, deadline pairs as simple constraints, and EDF based Total Bandwidth server [13] and constant bandwidth server [1] as runtime algorithms. The amount of desired flexibility can be set in this step as well.

Our transformation technique can extract maximum flexibility. By tightening start time and deadlines of certain tasks, it is possible to constrain the execution of some tasks, e.g., for reasons of testing or reliability.

Our method works by reducing complexity (NP hard in the case of distributed, precedence constrained executions with end-to-end deadlines) offline by instantiating a set of independent tasks with starttimes, deadlines constraints on single processors which fulfill application constraints and guaranteed bandwidth requirements. The issues of allocation to nodes, subtask deadline assignment, fulfilling jitter requirements are resolved by the offline scheduler. This allows the use of time intensive algorithms to resolve the constraints, since they are performed offline, i.e., before the system is deployed, and flexible scheduling at runtime.

Once tasks with starttime, deadline constraints have been derived and analyzed, earliest deadline first scheduling is performed on single nodes individually at runtime; the original set of complex constraints, distribution, etc., remains hidden from online scheduling.

The resulting instance of simple constraints will not generally be optimum. Since it is performed offline, however, additional analysis can be performed, possibly resulting in another instantiation with different simple constraints. Consider a subtask deadline assignment which induces tight constraint on one node. Performance analysis may show a different, more relaxed setting to be more appropriated.

B Offline schedule construction and bandwidth reservation strategies

As an additional requirement, the offline scheduler has to create a schedule such that a desired fraction U_s of the processor utilization (i.e., a desired bandwidth) is reserved for online aperiodic service. This means that, if a bandwidth U_s is reserved on a node, then for any interval $[t_1, t_2]$, there must be at least $(t_2 - t_1)U_s$ time available for aperiodic processing.

A trivial approach is to replace the worst case execution time of each task with $\frac{C}{1-U_S}$. No modifications to the scheduler are required to guarantee a bandwidth of U_S. This method, however, does not consider spare capacities in the schedule for bandwidth reservation. Response times in the resulting scheduling can thus be prohibitively long.

Our bandwidth reservation method during offline schedule construction analyzes the schedule for idle resources and their distribution. It maximizes flexibility by considering the leeway in the schedule, as per the specification constraints. In particular, the offline scheduler contains a function with tests the feasibility of the schedule constructed so far; it is extended by testing the availability of the specified bandwidth as well.

C Transformation technique

The feasible schedule with guaranteed bandwidth is transformed into independent tasks with starttimes, deadline pairs. Our method is based on the preparations for online scheduling in slot shifting [7].

The offline scheduler allocates tasks to nodes and resolves the precedence constraints. The scheduling tables list fixed start- and end times of task executions, that are less flexible than possible. The only assignments fixed by specification are starts of first and completion of last tasks in chains with end-to-end constrains, and tasks sending or receiving inter-node messages. The points in time of execution of all other tasks may vary within the precedence order. We calculate earliest start-times and latest finish-times for all tasks per node based on this knowledge. As we want to determine the maximum leeway of task executions, we calculate the deadlines to be as late as possible.

Let $end(PG)$ denote the end and $start(PG)$ the start of a precedence graph PG according to the schedule. The start of an inter-node message transmission M is denoted $start(M)$, the time it is available at all receiving nodes $end(M)$. These are the only fixed start times and deadlines, all others are calculated recursively with respect to precedence successors.

These fixed constraints are derived first: The *deadline* of task T, d_T, of precedence graph PG in a schedule is:
If T is exit task in PG: $d_T = dl(PG)$,
if T sends an inter-node message M: $d_T = start(M)$.

The *earliest start time* of task T, r_T, of precedence graph PG in a schedule is calculated in a similar way:
If T is entry task: $r_T = start(PG)$,
if T receives an inter-node message M: $r_T = end(M)$.

Next, constraints of predecessors and successors of tasks with fixed constraints are derived:
A predecessor P of a task T with fixed deadline is assigned a deadline so as to be executed before T with EDF, i.e.,
$d_P = d_T - C_T$.
A successor S of a task T with fixed starttime is assigned the same starttime as T. An appropriate deadline and EDF with ensure P preceding T.

[1] This serves as example; a number of other offline scheduling algorithm can be applied, e.g., the one presented in [10].

$r_P = r_T$.

This step is applied recursively.

D Online scheduling

Once the transformation is performed off line and a bandwidth U_s is reserved on each processing node, on line scheduling of aperiodic tasks can be handle by a Total Bandwidth Server (TBS). This service mechanism was proposed by Spuri and Buttazzo [12, 13] to improve the response time of soft aperiodic requests in a dynamic real-time environment, where tasks are scheduled according to EDF.

The server works as follows: whenever an aperiodic request enters the system, the total bandwidth (in terms of cpu execution time) of the server, is immediately assigned to it. This is done by simply assigning a suitable deadline to the request, which is scheduled according to the EDF algorithm together with the periodic tasks in the system. The assignment of the deadline is done in such a way to preserve the schedulability of the other tasks in the system.

In particular, when the k-th aperiodic request arrives at time $t = r_k$, it receives a deadline

$$d_k = \max(r_k, d_{k-1}) + \frac{C_k^a}{U_s},$$

where C_k^a is the execution time of the request and U_s is the server utilization factor (i.e., its bandwidth). By definition $d_0 = 0$. The request is then inserted into the ready queue of the system and scheduled by EDF, as any periodic or sporadic instance. Note that the maximum between r_k and d_{k-1} is needed to keep track of the bandwidth already assigned to previous requests.

Figure 1 shows an example of schedule produced with a TBS. The first aperiodic request, arrived at time $t = 6$, is assigned a deadline $d_1 = r_1 + \frac{C_1}{U_s} = 6 + \frac{1}{0.25} = 10$, and since d_1 is the earliest deadline in the system, the aperiodic activity is executed immediately. Similarly, the second request receives the deadline $d_2 = r_2 + \frac{C_2}{U_s} = 21$, but it is not serviced immediately, since at time $t = 13$ there is an active periodic task with a shorter deadline (18). Finally, the third aperiodic request, arrived at time $t = 18$, receives the deadline $d_3 = \max(r_3, d_2) + \frac{C_3}{U_s} = 21 + \frac{1}{0.25} = 25$ and is serviced at time $t = 22$.

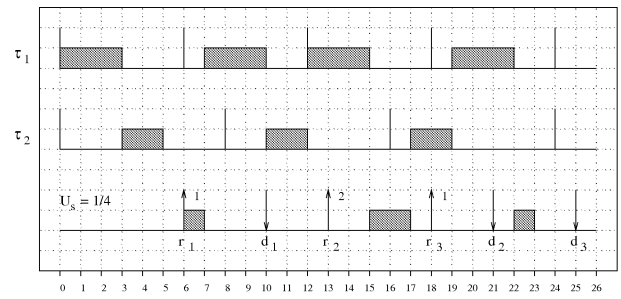

Figure 1: Total Bandwidth Server example.

Intuitively, the assignment of the deadlines is such that in each interval of time, the fraction of processor time allocated by EDF to aperiodic requests never exceeds the server utilization U_s. Since the processor utilization due to aperiodic tasks is at most U_s, the schedulability of a periodic task set in the presence of a TBS can simply be tested by verifying the following condition:

$$U_p + U_s \leq 1,$$

where U_p is the utilization factor of the periodic task set. This results is proved by the following theorem.

Theorem 1 (Spuri and Buttazzo, 96). *Given a set of n periodic tasks with processor utilization U_p and a TBS with processor utilization U_s, the whole set is schedulable* if *and* only if

$$U_p + U_s \leq 1.$$

The implementation of the TBS is straightforward, since to correctly assign the deadline to a new request, we only need to keep track of the deadline assigned to the last aperiodic request (d_{k-1}). Then, the request can be inserted into the ready queue and treated by EDF as any other periodic instance. Hence, the overhead is practically negligible.

IV Example

In this section, we will illustrate our methods with an example. The system consists of two processing nodes; for the simplicity of the example, we assume that the sending of a message takes one time unit. There are 7 tasks to be offline scheduled: A, B, C, D, E, Y, Z, $A - E$ form a precedence constrained execution chain, with the following precedence constraints: $A \to B, B \to C, B \to E, C \to D, C \to E$. The global precedence graph is shown in Figure 2.

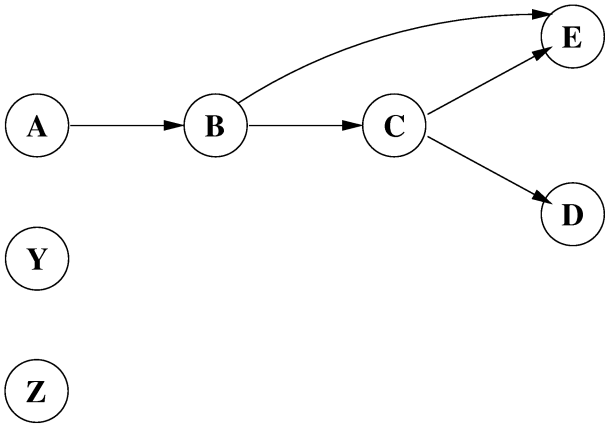

Figure 2: Precedence graph for the tasks of the example.

Tasks A, B, E are allocated to node 0 and C, D to 1. Task A has a jitter requirement: the variation in execution has to be less than or equal 1 for two succeeding instances. The starttime of the execution chain is 0, the end-to-end deadline, i.e., the maximum time interval between $start(A)$ and $max(end(D), end(E)$ 11. Y, allocated to node 0, has starttime of 4 and deadline 9, and Z on node 1 of 0 and 6, resp. Worst case execution times: $A = 2, B = 1, C = 1, D = 1, E = 1, Y = 2, Z = 2$. The required bandwidth $U_S = \frac{1}{3}$.

A *Transformation into simple constraints*

The task schedule together with the original constraints is transformed into independent tasks on single nodes with start time, deadline pairs (denoted as r_i, d_i). First, it identifies constraints due to end-to-end, internode communication, and jitter:

End-to-end constraints:

$r_A = 0, d_E = d_D = 11$.

Internode communication: $d_B = 5, r_C = 6, d_C = 8, r_E = 9$.

Jitter: $d_A = r_A + wcet(A) - jitter = 3$.

Next, constraints for successors and predecessors of these tasks are derived:

$r_B = r_A = 0, r_D = r_C = 6$.

A, B cannot start before the starttime of the execution chain, thus they are assigned the same deadline. Since we use EDF for online scheduling, $d_A < d_B$ ensures the precedence order.

Y, Z are constrained by their original constraints:

$r_Y = 4, d_Y = 9; r_Z = 0, d_Z = 6$.

Table 3 summarizes the derived constraints.

task	r_i	d_i
A	0	3
B	0	5
C	6	8
D	6	11
E	9	11
Y	4	9
Z	0	6

Figure 3: Derived simple constraints.

These simple constraints comprise the original constraints as per the offline constructed schedule and guarantee a bandwidth $U_S = \frac{1}{3}$.

The resulting EDF schedule on the transformed task set is depicted in Figure 4.

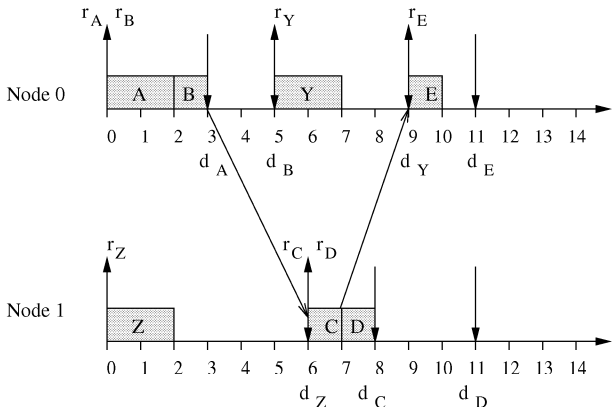

Figure 4: Schedule produced by EDF on the offline transformed task set.

B *Online Scheduling*

The available bandwidth $U_s = 1/3$ reserved by the offline algorithm, can be exploited by a Total Bandwidth Server (TBS) to efficiently handle online aperiodic requests. In the example, a request J_1 with computation time $C_1 = 1$ arrives on Node 0 at time $t = 1$. Hence, the TBS assigns it a deadline $d_1 = t + C_1/U_s = 4$. As a consequence, J_1 is executed before task B, since $d_1 < d_B$.

At time $t = 5$, another request J_2 with computation time $C_2 = 2$ arrives on Node 0. This time, the request is assigned a deadline $d_2 = t + C_2/U_s = 11$, which is the same as the deadline of task E. However, since deadline ties are broken in favor of the server, J_2 executes before E.

On Node 1, a request J_3 with computation time $C_3 = 2$ arrives at time $t = 1$. Hence, according to the TBS rule, it is assigned a deadline $d_3 = t + C_3/U_s = 7$. At time $t = 5$, another request J_4 arrives before deadline d_3. In this case, the TBS rule states that the deadline has to be computed as $d_4 = max(t, d_3) + C_2/U_s = 10$, since the bandwidth U_s was already assigned to J_3 up to time d_3. As a result, since $d_4 < d_D$, the execution of J_4 precedes that of task D.

The integrated schedule is shown in Figure 5

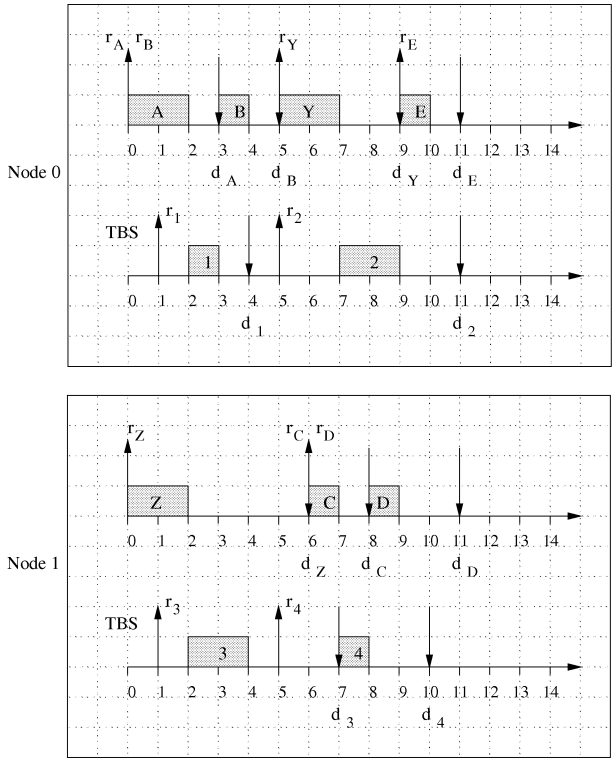

Figure 5: Schedule produce by EDF using the integrated approach.

V Simulations

We implemented and simulated the described algorithm. All the offline and soft aperiodic tasks were randomly generated, and the load of the offline tasks is varied between 0 and

0.9, and the soft aperiodic task load is varied between three different levels over the lcm.

The simulation length for each run was 10000 slots, and the soft aperiodic tasks arrived during that length.

We have studied the average response times of the soft aperiodic tasks, and compared our algorithm against background scheduling. We also check what happens with the soft tasks when the offline load is increased. When the offline load is equal to 0 only TBS is running.

Figure 6 shows the result of the simulation. The same task sets were run with both background scheduling and our method, and the response times are measured in slots.

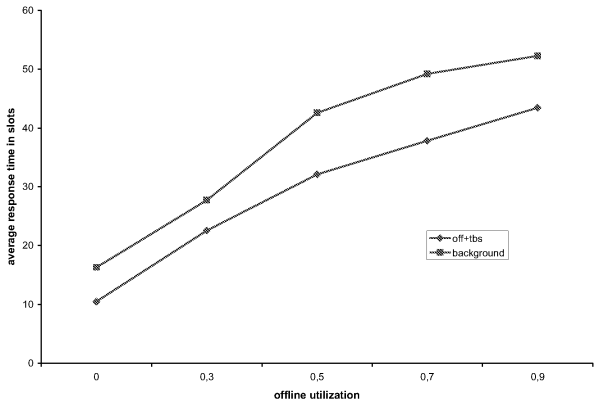

Figure 6: Response times for the soft aperiodic tasks.

Due to high offline load, many soft aperiodic tasks did not finish before the simulation ended and therefore the response times became higher with increased load. Even when the offline load was 0 some soft aperiodic tasks were not able to finish because the ready queue was overloaded.

VI Conclusion

In this paper, we presented a method to efficiently handle soft aperiodic tasks in offline scheduled systems, to handle real-world constraints such as distribution, end-to-end deadlines, jitter control, periodic and non periodic tasks, as well as efficiency, cost effectiveness and more.

We propose using an offline scheduler to handle complex constraints, reduce their complexity, and provide for guaranteed available bandwidth. In order to apply standard earliest deadline first scheduling during system operation, we presented a technique, which transforms the schedule and original constraints into independent tasks with starttime, deadline constraints on single nodes only. Consequently, the runtime mechanisms are very simple, flexible, and efficient.

We have shown how these resulting simple constraints can be used by the standard total bandwidth server algorithm for flexible and efficient execution, resource reclaiming, and overload handling.

Instead of different feasibility tests for different types of constraints, our method caters for a variety of constraints with only minor modifications in the offline scheduler. Since changes and additions to the set of constraints can be incorporated into an offline scheduler relatively easy, our method also provides for variations and modifications in task constraints, as, e.g., induced by industrial design processes and system life cycles.

References

[1] L. Abeni and G. Buttazzo,"Integrating Multimedia Applications in Hard Real-Time Systems", *In Proceedings of the IEEE Real-Time Systems Symposium*, Dec. 1998.

[2] H. Chetto and M. Chetto, "Some Results on the Earliest Deadline Scheduling Algorithm", *IEEE Transactions on Software Engineering*, 15(10), Oct. 1989.

[3] R.I. Davis, K.W. Tindell, and A. Burns. Scheduling slack time in fixed priority pre-emptive systems. *In Proceedings of the Real-Time Symposium*, Dec. 1993.

[4] Z. Den, J. W. S. Liu, and J. Sun, "A Scheme for Scheduling Hard Real-Time Applications in Open System Environment", *In Proceedings of the 9-th Euromicro Workshop on Real-Time Systems*, June 1997.

[5] Z. Deng and J. W. S. Liu, "Scheduling Real-Time Applications in Open Environment", *In Proceedings of IEEE Real-Time System Symposium*, Dec. 1997.

[6] G. Fohler. Analyzing a pre run-time scheduler and precedence graphs. Research Report 13/92, Institut für Technische Informatik, Technische Universität Wien, Sep. 1992.

[7] G. Fohler. Joint scheduling of distributed complex periodic and hard aperiodic tasks in statically scheduled systems. *In Proceedings Real-Time Systems Symposium*, Dec. 1995.

[8] H. Kopetz, G. Fohler, G. Grünsteidl, H. Kantz, G. Pospischil, P. Puschner, J. Reisinger, R. Schlatterbeck, W. Schütz, A. Vrchoticky, and R. Zainlinger. The distributed, fault-tolerant real-time operating system MARS. *IEEE Operating Systems Newsletter*, 6(1), 1992.

[9] C. W. Mercer, S. Savage, and H. Tokuda, "Processor Capacity Reserves for Multimedia Operating Systems" *In Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, May 1994.

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

[11] K. Ramamritham, G. Fohler, and J.-M. Adan. Issues in the static allocation and scheduling of complex periodic tasks. *In Proceedings 10th IEEE Workshop on Real-Time Operating Systems and Software*, May 1993.

[12] M. Spuri and G.C. Buttazzo, "Efficient Aperiodic Service under Earliest Deadline Scheduling", *In Proceedings of IEEE Real-Time System Symposium*, Dec. 1994.

[13] M. Spuri and G.C. Buttazzo, "Scheduling Aperiodic Tasks in Dynamic Priority Systems," *In Real-Time Systems*, 10(2), Dec. 1996.

[14] J. A. Stankovic and K. Ramamritham. The Spring kernel: A new paradigm for real-time operating systems. *In IEEE Software*, May 1991.

[15] S. R. Thuel and J.P. Lehoczky. Algorithms for scheduling hard aperiodic tasks in fixed-priority systems using slack stealing. *In Proceedings of the Real-Time Symposium*, Dec. 1994.

[16] W. Zhao, K. Ramamritham, and J. A. Stankovic. Preemptive scheduling under time and resource constraints. *In IEEE Transactions on Computers*, C-36(8):949–960, Aug. 1987.