Non-Preemptive Scheduling with History-Dependent Execution Time

Björn Andersson Sagar Chaki Dionisio de Niz Brian Dougherty Russell Kegley Jules White
baandersson@sei.cmu.edu chaki@sei.cmu.edu dionisio@sei.cmu.edu
brian.dougherty.work@gmail.com russell.b.kegley@lmco.com jules.white@gmail.com

***Abstract*—Consider non-preemptive fixed-priority scheduling of arbitrary-deadline sporadic tasks on a single processor assuming that the execution time of a job J depends on the actual schedule (sequence) of jobs executed before J. We present exact schedulability analysis for such a system.**

I. Introduction

The trend in computer architecture during the recent decades has been towards processors with state that does not impact the computed result but improves performance. For example, cache memories make the time required for executing a memory instruction (e.g. load or store) dependent on previously executed memory instructions (potentially from a job of another task). This makes the execution time of a job dependent on jobs that executed before it — as noted in a recent study [1]. Hence, there is a need to develop scheduling theory where the maximum execution time of a job is not described by a single number but the maximum execution time is described by a richer model. This model should state that if scheduling is done in a certain way then the maximum execution time of a given job will be a pre-specified value.

The real-time systems research community has created models describing (i) the possible job arrivals, (ii) that deadlines of different jobs of the same task can be different and (iii) that execution times of jobs of the same task can be different. Some theories [2]–[7] analyze the possible jobs directly whereas others [8], [9] compute an event stream for each task and perform schedulability analysis based on event streams. Some works (e.g. [10] and the later part of [9] discussing inter-event streams) use models where the arrival of a job is dependent on the arrival of a job of another task. Common to these works, however, is that although some of them can describe how the execution time of a job J depends on other jobs that executed before J, they cannot describe that the execution time of job J depends on the actual schedule (sequence) of jobs executed before J. To see why this is problematic for modeling cache effects, consider a set of three jobs, $\{J,J',J''\}$, scheduled non-preemptively where job J shares working set with job J' but job J does not share working set with job J''. In this case, if the processor executes jobs in the sequence $<J'',J',J>$ then the upper bound on the execution time of job J should be lower than if the processor executes jobs in the sequence $<J',J'',J>$. The reason is that with the former sequence, $<J'',J',J>$, it holds that immediately after J' has executed, some of J's working set is already in the cache. In order to properly model this effect, the execution time should be described so that the execution time of a job depends on the actual schedule (sequence) of jobs that executed before it. Unfortunately, the current research literature in real-time systems does not offer this.

Therefore, in this paper, we present a new model for describing non-preemptive fixed-priority scheduling where the execution time of a job depends on the actual schedule (sequence) of jobs executed before it. We show that exact schedulability analysis in this model is NP-complete in the strong sense and hence we should not expect to perform exact schedulability analysis with pseudo-polynomial time-complexity. For this reason, we present an exact schedulability test which computes response times of tasks based on solving Mixed-Integer Linear Programs (MILP).

The remainder of this paper is structured as follows. Section II presents the system model. Section III reasons about response times and defines quantities which we need for computing the response time. Section IV presents an optimization problem whose solution is one of these quantities. Section V presents another such quantity. Section VI shows how to put these two quantities together into an algorithm for computing response times. Section VII discusses computational complexity. Section VIII gives conclusions.

II. System Model

A. Task and platform characterization

Consider a task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ scheduled on a single processor. A task $\tau_i \in \tau$ has two associated positive real numbers D_i and T_i with the interpretation that the task generates a (potentially infinite) sequence of jobs where the arrival times of jobs generated by τ_i are separated by at least T_i time units and each job of task τ_i must finish its execution within D_i time units after its arrival.

It is common to categorize a task set as either (i) implicit-deadline task set (where $\forall i : D_i = T_i$), (ii) constrained-deadline task set (where $\forall i : D_i \leq T_i$) or (iii) arbitrary-deadline task set (where $\forall i : D_i$ can take any positive value; specifically, it is allowed that $D_i > T_i$). We consider arbitrary-deadline sporadic tasks.

We assume that the execution time of a job J depends on the actual schedule (sequence) of jobs executed before J. For specifying the execution time, Figure 1(a) presents static task parameters and Figure 1(b) specifies how bounds

Variable	Type	Defined for	Interpretation
$nhlbc_i$	integer$\geq$1	$i \in 1..n$	number of pre-specified histories for specifying lower bound on exec. time of τ_i
$nhubc_i$	integer$\geq$1	$i \in 1..n$	number of pre-specified histories for specifying upper bound on exec. time of τ_i
$lhlbc_i^h$	integer$\geq$0	$i \in 1..n$, $h \in 1..nhlbc_i$	length of h^{th} pre-specified history used to specify lower bound on exec. time of τ_i
$lhubc_i^h$	integer$\geq$0	$i \in 1..n$, $h \in 1..nhubc_i$	length of h^{th} pre-specified history used to specify upper bound on exec. time of τ_i
$ihlbc_i^{h,k}$	integer$\in 1..n$	$i \in 1..n$, $h \in 1..nhlbc_i$, $k \in 1..lhlbc_i^h$	k^{th} item in h^{th} pre-specified history used to specify lower bound on exec. time of task τ_i
$ihubc_i^{h,k}$	integer$\in 1..n$	$i \in 1..n$, $h \in 1..nhubc_i$, $k \in 1..lhubc_i^h$	k^{th} item in h^{th} pre-specified history used to specify upper bound on exec. time of task τ_i
lbc_i^h	real number$\geq$0	$i \in 1..n$	lower bound on exec. time of τ_i if h^{th} pre-specified history is matched
ubc_i^h	real number$\geq$0	$i \in 1..n$	upper bound on exec. time of τ_i if h^{th} pre-specified history is matched

(a) Symbols used to define pre-specified histories and corresponding execution times of tasks. It is assumed that for each task τ_i there is one h such that $lhlbc_i^h = 0$ and there is one h such that $lhubc_i^h = 0$.

$$hmlbc(h, J, S) = \left(lhlbc^h_{task(J)} \leq npred(J, S)\right) \ \wedge \left(\bigwedge_{k=1}^{lhlbc^h_{task(J)}} ihlbc^{h,k}_{task(J)} = task(pred(J, S, k))\right)$$

$$hmubc(h, J, S) = \left(lhubc^h_{task(J)} \leq npred(J, S)\right) \ \wedge \left(\bigwedge_{k=1}^{lhubc^h_{task(J)}} ihubc^{h,k}_{task(J)} = task(pred(J, S, k))\right)$$

$$hmsetlbc(J, S) = \{h : 1 \leq h \leq nhlbc_{task(J)} \wedge hmlbc(h, J, S)\}$$

$$hmsetubc(J, S) = \{h : 1 \leq h \leq nhubc_{task(J)} \wedge hmubc(h, J, S)\}$$

$$lbcj_J(S) = \max_{h \in hmsetlbc(J,S)} lbc^h_{task(J)}$$

$$ubcj_J(S) = \min_{h \in hmsetubc(J,S)} ubc^h_{task(J)}$$

The execution time of job J is a real number in the range $[\, lbcj_J(S), ubcj_J(S) \,]$

(b) Functions defining how the execution time of a job J depends on pre-specified histories and the schedule S at run-time.

$pred(J, S, k)$ denotes the k:th job executed before job J in schedule S.

$npred(J, S)$ denotes the maximum integer such that from the starting time of the execution of job $pred(J, S, npred(J, S))$ until the finishing time of the execution of job J, it holds that the processor is busy.

$task(J)$ denotes the index of the task that generated job J.

(c) Auxiliary functions.

Figure 1. Defining how the execution time of a job depends on jobs that executed before it.

on the execution time of a job is obtained from these task parameters. Intuitively, each task is specified with many upper bounds on the execution time and each of these upper bounds has an associated sequence of jobs (which we call pre-specified history). For a job J in a schedule S, the jobs executed before J in S are compared against the pre-specified histories of the task that generated J. If a pre-specified history matches S then the upper bound associated with the pre-specified history is an upper bound on the execution time of J. It may happen that more than one pre-specified history matches and hence J may have more than one upper bound on its execution time. If so, all of them apply and hence, the lowest among them is chosen as a single upper bound. The lower bound on the execution time of a job is defined analogously.

Figure 2(a) shows an example illustrating the model and how the execution time depends on history. Executing a job of τ_1 just before a job of τ_2 reduces the execution time of the job of τ_2, and vice versa. The execution time of a job of τ_3 does not depend on other jobs though.

B. Scheduling

We assume tasks are assigned unique priorities and each job generated by task τ_i is given the priority of τ_i. Let $hp(i)$ denote the set of tasks with higher priority than task τ_i. We say that a job J is eligible at time t, if (i) job J arrives at t or earlier and (ii) job J finishes execution later than t.

We assume work-conserving scheduling, that is, the processor cannot be idle at time t if there is an eligible task at time t. We assume non-preemptive scheduling, that is, if a job has started to execute then it continues to execute

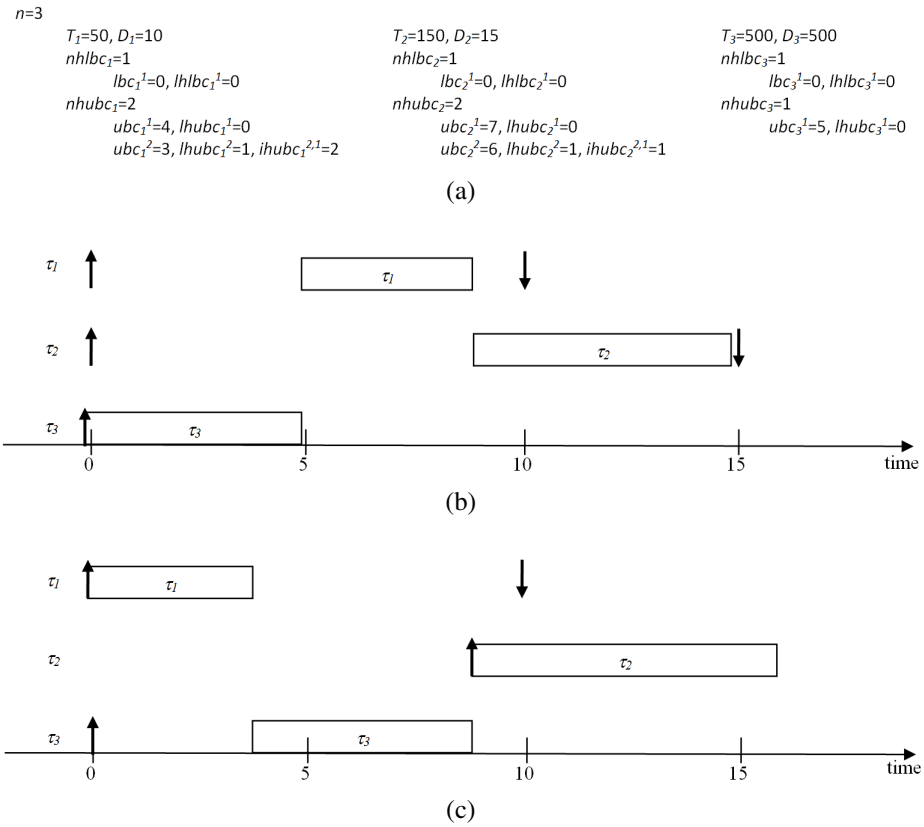

Figure 2. (a) An example illustrating the model and how the execution time depends on history. An arrow pointing upwards indicates the arrival of a job. An arrow pointing downwards indicates the deadline of a job. In this example, τ_1 has higher priority than task τ_2 and task τ_2 has higher priority than task τ_3. Executing a job of τ_1 just before a job of τ_2 reduces the execution time of the job of τ_2, and vice versa. This can be seen by the fact that the upper bound on the execution time (ubc) of task τ_2 is seven ($ubc_2^1 = 7$) if we know nothing about the jobs that executed before it ($lhubc_2^1 = 0$) but it is six ($ubc_2^2 = 6$) if we know that it executed after a job of task τ_1 ($ihubc_2^{2,1} = 1$). The tasks have lower bounds on execution times as well but these are set to zero ($lbc = 0$); (b) A schedule generated for a specific arrival pattern for the task set in (a); (c) A schedule generated for another specific arrival pattern for the task set in (a).

until it finishes. When a job finishes, the job selected for execution is the one with the highest priority among the jobs that are eligible at that time. Figure 2(b) and Figure 2(c) show examples of schedules. In case two jobs of the same task are eligible at the same time (this can happen for a task with $D_j > T_j$), jobs of such a task are served in FIFO order.

We assume that the task set is such that for each possible assignment of job arrival times that satisfy the constraint of minimum inter-arrival times it holds that the processor cannot be busy forever. This assumption is analogous to the assumption that the utilization is strictly less than 1 – an assumption used in other models, such as the normal sporadic model where the (worst-case) execution time of a job does not depend on the jobs that executed before it. For this purpose, we define $\widehat{U}$ and $\breve{U}$ as follows:

$$\widehat{U} = \sum_{j=1}^{n} \frac{\max_{h=1}^{nhubc_j} ubc_j^h}{T_j}$$

$$\breve{U} = \sum_{j=1}^{n} \frac{\min_{h=1}^{nhubc_j} ubc_j^h}{T_j}$$

If $\widehat{U} < 1$ then the processor cannot be busy forever. If $\breve{U} > 1$ then the processor will be busy forever if for each job J the execution time is $ubcj_J(S)$.

C. Response-time and schedulability

The response time of a job is the time that the job finishes execution minus the arrival time of the job. The response time of a task τ_i (denoted R_i) is the maximum response time that a job of τ_i can experience. A task set is schedulable with respect to a given priority assignment if $\forall i : R_i \leq D_i$.

Note that in Figure 2(b), the job of task τ_2 has execution time of six time units because it executes after a job of task τ_1. Hence the job of task τ_2 meets its deadline. But with previously known non-preemptive analysis [11], [12], which does not consider that the execution time of a job depends on the jobs that execute before it, the calculated response time for task τ_2, would be one time unit longer than the one in the example in Figure 2(b) and hence with previously known non-preemptive analysis, the task set in Figure 2(a) would be deemed unschedulable.

Figure 2(c) shows an example of a schedule for another arrival pattern. Note here that the job of τ_2 does not execute directly after a job of task τ_1 and hence the execution time of the job of task τ_2 is seven, that is, one time unit more than it was in Figure 2(b).

III. Reasoning About Response Time

Recall that our goal is to develop an algorithm for computing R_i – the response time of task τ_i. For this purpose, we will reason about a necessary condition for the situation where a job of τ_i experiences the maximum response time. This will lead us to optimization problems (which we present in Sections IV and V). And then we present the entire new algorithm (in Section VI) for computing R_i.

Previous works on computing the response time of tasks scheduled with fixed-priority scheduling have relied on two ideas: (i) the response time of a job of task τ_i is maximized if it arrives at its critical instant, i.e., when a job of task τ_i arrives at a time when a job from each task in $hp(i)$ arrives [13]–[15] and (ii) the response time of a job of task τ_i is maximized if the job arrives in a level-i busy period, that is, a time interval where only jobs of tasks of priority i or higher starts execution [11], [12].

The former idea applies to preemptive fixed-priority scheduling but not to non-preemptive fixed-priority scheduling. Also, for some special cases (arbitrary-deadline tasks [16] or subtasks [17]) it cannot be used even for preemptive scheduling. The second idea applies to non-preemptive scheduling and to preemptive scheduling of the two special cases mentioned above but not to non-preemptive scheduling where the execution time of a job depends on the actual schedule (sequence) of jobs executed before it (since the execution time of the first job of the level-i busy period depends on the job before it and this job is outside the level-i busy period). Therefore, to develop a schedulability analysis for our model – with non-preemptive scheduling of jobs where the execution time of a job depends on the actual schedule (sequence) of jobs executed before it – we will instead use another, previously known, concept: *busy period*.

We define a busy period as a time interval [ts,te) for which the following holds:

1) It starts at some time ts when a job is queued ready for execution and there are no jobs waiting to be executed that were queued strictly before time ts.
2) It is a contiguous interval of time during which the processor is busy.
3) It ends at the earliest time te when the processor becomes idle and $te > ts$.

From the above definition of busy period, we observe:

O1. For a schedule generated by a given task set, arrival time assignment and priority assignment, it holds that if there is more than one busy period in the schedule then these busy periods are non-overlapping in time.

If the processor was idle just before a job J arrived then a new busy period starts at the time of the arrival of job J. Clearly, this also holds for the case that J is the job that has the maximum response time among all jobs generated by task $task(J)$. This gives us the observation:

O2. For a schedule generated by a given task set, arrival time assignment and priority assignment, it holds for the job of τ_i with the largest response time of all jobs of τ_i, its arrival time is in a busy period.

From O1 and O2, it follows that in order to compute the response time of a task, we only need to consider a schedule with a single busy period.

In real-time scheduling where the execution time does not depend on history, it holds that the duration of a busy period depends on the number of jobs and their execution times but it does not depend on the sequence in which these jobs execute. Regarding the duration of busy period in our model, however, we observe:

O3. Consider a set of jobs. The duration of the busy period resulting from one sequence of these jobs may differ from the duration of the busy period resulting from another sequence of these jobs.

This observation is relevant because although in schedulability analysis, the task set is fixed and the priority assignment is fixed, the actual sequence of jobs generated at run-time depend on arrival times of jobs and these are not specified offline.

Let L denote the maximum duration that a busy period can take. We can clearly compute Q_i, an upper bound on the number of jobs from task τ_i that could possibly execute in a busy period, as:

$$Q_i = \left\lceil \frac{L}{T_i} \right\rceil \tag{1}$$

Let $\tau_{i,q}$ denote the q:th job from τ_i arriving in the busy period. Also, let $R_{i,q}$ denote the maximum response time of $\tau_{i,q}$. Considering once again that execution time is history dependent, we observe:

O4. Consider a set of jobs where $\tau_{i,q}$ is one of the jobs. The sequence (formed from these jobs) that maximizes the duration of the busy period is not necessarily the same sequence (formed form the same set of jobs) that maximizes $R_{i,q}$.

Let $A_{i,q}$ denote the arrival time of $\tau_{i,q}$ and let $f_{i,q}$ denote the finishing time of $\tau_{i,q}$. Let $valid_{i,q}$ be true if there exist an assignment of arrival times such that for the resulting sequence of jobs, which in turn creates a busy period (not necessarily the one of maximum duration), it holds that there are at least q jobs of task τ_i in the busy period.

For the case that the execution times do not depend on history, we obtain that $\forall q \in [1, Q_i]$, it holds that $valid_{i,q}$ is true. But for our model, we observe:

O5. There is a task set with a task τ_i for which there is a $q \in [1, Q_i]$ such that $valid_{i,q}$ is false.

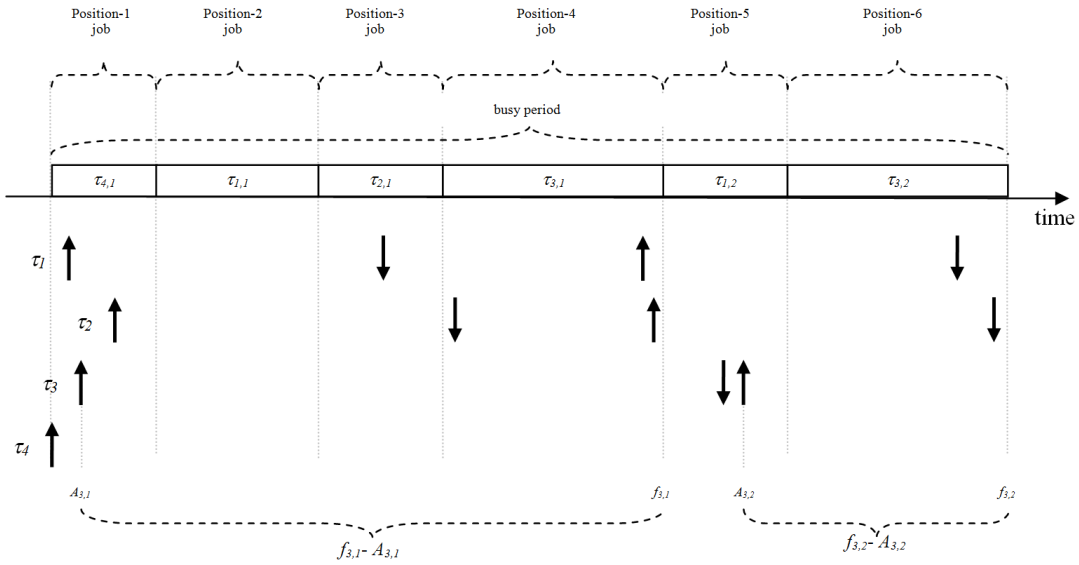

Figure 3. Illustration of the concepts $\tau_{i,k}$, $f_{i,q}$, $A_{i,q}$ and position-p job applied on a task set where we want to compute the response time of τ_3.

Based on these observations, our approach to compute R_i is as follows: (i) compute L, (ii) compute Q_i, (iii) for each $q \in [1, Q_i]$: compute $valid_{i,q}$ and $R_{i,q}$ and (iv) compute $R_i = \max_{q \in [1,Q_i] \wedge valid_{i,q}} R_{i,q}$.

Figure 3 illustrates concepts.

In order to turn this approach into an algorithm, it is necessary to specify a method for computing L, $valid_{i,q}$ and $R_{i,q}$. We will do so using two functions.

We define $requested(t)$ as a function returning the maximum duration that the processor can be busy from executing jobs that arrive in a time interval of duration t such that just before this time interval the processor was idle. It also returns a sequence of jobs corresponding to this duration returned. One can show that solving $< t$, $seq >$ = $requested(t)$ and taking the solution t gives us the maximum duration of a busy period. We will compute $requested(t)$ by solving an optimization problem where binary variables represent the sequence of jobs in a busy period and maximize the cumulative execution time of these jobs considering the history dependency of execution times.

We define $resp(i, q, t)$ as a function which returns $valid_{i,q}$ and $R_{i,q}$. Internally, $resp(i, q, t)$ solves an optimization problem similar to the one in $requested(t)$ but with $f_{i,q} - A_{i,q}$ as objective function. If the optimization problem is feasible then $valid_{i,q}$ is true; otherwise false.

IV. Computing $requested(t)$

We will now define the function $requested(t)$ as a solution to an optimization problem. We know that the task set, together with arrival times of jobs and the priority assignment results in a schedule and this schedule can be represented as a sequence of positions where a position is a time interval such that no context switch occurs in this time interval. We will introduce decision variables describing which task executes in a given position. In order to solve an optimization problem, an upper bound on the number of variables is needed and hence we need an upper bound on the number of positions. Since t is given as input, an upper bound on the number of positions is:

$$maxp = \sum_{j \in \{1,2,\ldots,n\}} \left\lceil \frac{t}{T_j} \right\rceil \tag{2}$$

Let $njobs_j$ denote an upper bound on the number of jobs of task τ_j that arrive during a time interval of duration t.

$$\forall j \in \{1, 2, \ldots, n\} : njobs_j = \left\lceil \frac{t}{T_j} \right\rceil \tag{3}$$

We are interested in exploring all possible schedules that can be generated on positions 1, 2, ..., $maxp$ subject to the constraints from the task parameters and dispatching rules. We must be able to explore the case that the number of jobs from task τ_j executing in positions 1, 2, ..., $maxp$ is less than $njobs_j$ (this can happen because in the sporadic model, the time between arrivals of two consecutive jobs of task τ_j can be greater than T_j) and we will therefore introduce a special position position $maxp + 1$ which hosts all jobs that are not executing in the positions 1, 2, ..., $maxp$.

Let t_p denote the starting time of position p. Let $ftlastjob$ denote the last time for which it holds that the processor is busy during [t_1,$ftlastjob$) and the processor is idle during [$ftlastjob$,t_{maxp+1}). Let the objective function of our optimization problem be maximize $ftlastjob - t_1$ subject to certain constraints. We will therefore state these constraints and also introduce additional variables that are needed to express the constraints. Some of the constraints use logical

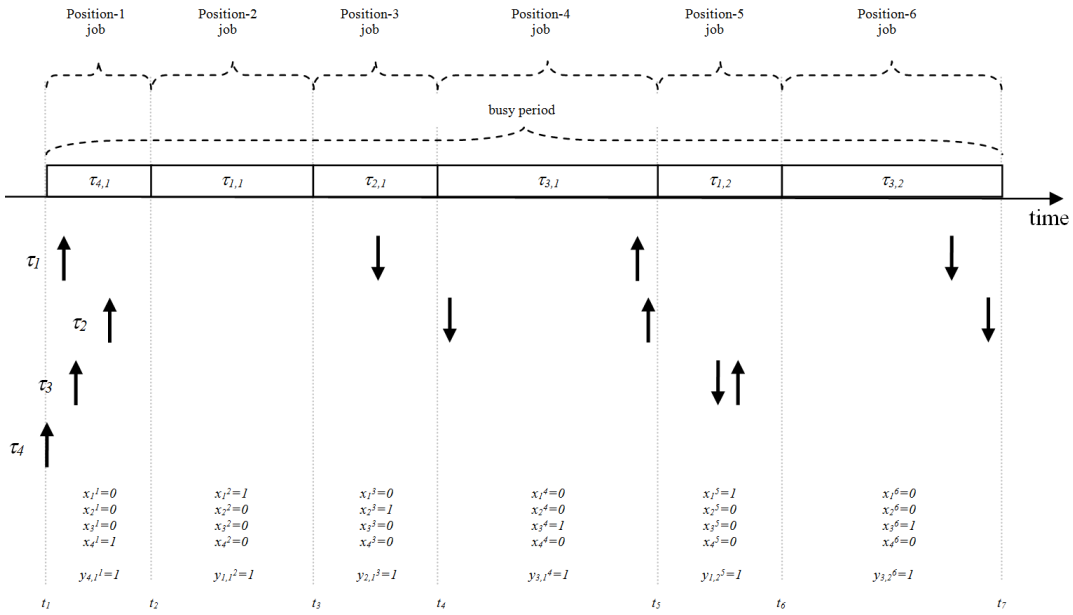

Figure 4. Illustration of how the variables x and t represent the schedule during a busy period.

operators which are not accepted in normal MILP solvers. They can be translated into MILP expressions using standard techniques though.

A. Representation of schedule

Let x_j^p be a variable being one if a job of task τ_j executes in position p; otherwise zero. Let $y_{j,k}^p$ be a variable being one if $\tau_{j,k}$ executes in position p; otherwise zero. See Figure 4.

From the definition of x_j^p and $y_{j,k}^p$, it clearly holds that:

$$\forall j \in [1,n], p \in [1, maxp] : x_j^p = \sum_{k=1}^{njobs_j} y_{j,k}^p \tag{4}$$

A specific job executes in exactly one position. Hence:

$$\forall j \in [1,n], k \in [1, njobs_j] : \textstyle\sum_{p=1}^{maxp+1} y_{j,k}^p = 1 \tag{5}$$

B. Arrival, finishing and dispatching

Let position 0 be the time at or before t_1. No job will execute there but this position turns out to be convenient for expressing job arrivals. Let $A_{j,k}$ denote the arrival time of $\tau_{j,k}$. Let $arrives_{j,k}^p$ be a variable indicating whether $\tau_{j,k}$ arrives in position p. Also, we require that a job arrives in exactly one of these positions. We have:

$$\begin{aligned} &\forall j \in [1,n], k \in [1, njobs_j], p \in [1, maxp] : \\ &\quad arrives_{j,k}^p = 1 \Rightarrow (t_p \leq A_{j,k} \wedge A_{j,k} \leq t_{p+1}) \end{aligned} \tag{6}$$

$$\begin{aligned} &\forall j \in [1,n], k \in [1, njobs_j] : \\ &\quad arrives_{j,k}^0 = 1 \Rightarrow A_{j,k} = t_1 \end{aligned} \tag{7}$$

$$\begin{aligned} &\forall j \in [1,n], k \in [1, njobs_j] : \\ &\quad arrives_{j,k}^{maxp+1} = 1 \Rightarrow A_{j,k} \geq t_{maxp+1} \end{aligned} \tag{8}$$

and

$$\forall j \in [1,n], k \in [1, njobs_j] : \sum_{p=0}^{maxp+1} arrives_{j,k}^p = 1 \tag{9}$$

Let $hasarrived_{j,k}^p$ be a variable being one if $\tau_{j,k}$ arrives at or before t_p.

$$\begin{aligned} &\forall j \in [1,n], k \in [1, njobs_j], p \in [1, maxp] : \\ &\quad hasarrived_{j,k}^p = \textstyle\sum_{p' \in [1,p-1]} arrives_{j,k}^{p'} \end{aligned} \tag{10}$$

Because tasks have minimum inter-arrival times, we have:

$$\forall j \in [1,n], k \in [1, njobs_j - 1] : A_{j,k+1} - A_{j,k} \geq T_j \tag{11}$$

Let $hasfinished_{j,k}^p$ be a variable being one if the k:th job of task τ_j has finished execution at or before the starting time of the position p; otherwise zero. Clearly, we have:

$$\begin{aligned} &\forall j \in [1,n], k \in [1, njobs_j], p \in [1, maxp] : \\ &\quad hasfinished_{j,k}^p = \textstyle\sum_{p' \in [1,p-1]} y_{j,k}^{p'} \end{aligned} \tag{12}$$

Let $rdy_{j,k}^p$ be a variable being one if the k:th job of task τ_j is ready at the starting time of position p, that is, it has arrived but not finished execution at or before position p has started; otherwise zero. Clearly, we have:

$$\begin{aligned} &\forall j \in [1,n], k \in [1, njobs_j], p \in [1, maxp] : (rdy_{j,k}^p = 1) \\ &\iff (hasarrived_{j,k}^p = 1) \wedge (hasfinished_{j,k}^p = 0) \end{aligned} \tag{13}$$

We can now express dispatching. Our assumption of non-preemptive fixed-priority scheduling gives us:

$$\begin{aligned}&\forall j \in [1,n], k \in [1, njobs_j], p \in [1, maxp]: \\ &\quad (rdy^p_{j,k} = 1 \wedge \\ &\quad (\forall j' \in hp(j), k' \in [1, njobs_{j'}] : rdy^p_{j',k'} = 0) \wedge \\ &\quad (\forall k' \in [1, k-1] : rdy^p_{j,k'} = 0)) \iff y^p_{j,k} = 1\end{aligned} \tag{14}$$

Note that a job is allowed to arrive in any position (positions [0,$maxp$+1]) but it is not allowed to execute in position 0 and hence the range over which the summation is taken in (5) is different from the one in (9). Also, note that (6) contains a non-strict inequality ($A_{j,k} \leq t_{p+1}$) and therefore, one may wonder whether it can happen that for a fixed j and k there is a p such that $arrives^p_{j,k} = 1$ and $arrives^{p+1}_{j,k} = 1$ which would be problematic. Note that (9) ensures that this cannot happen. Furthermore, note that (7) states that if a job arrives in position 0 then it must have been that the job arrived at time t_1. This is because the busy period started at time t_1 so whatever jobs executed before t_1 has no impact on the schedule that we are considering.

C. Expressing history-dependent execution time

Let us now discuss the execution time of a job. Let $zubc^{p,h}_j$ be a variable being one if a job of task τ_j executes in position p and ubc^h_j should be used as the tightest upper bound on the execution time for this job; otherwise zero. Let $zlbc^{p,h}_j$ be a variable being one if a job of task τ_j executes in position p and lbc^h_j should be used as the tightest lower bound on the execution time for this job; otherwise zero.

$$\begin{aligned}&\forall j \in [1,n], p \in [1, maxp+1]: \\ &\quad x^p_j = \textstyle\sum_{h=1}^{nhlbc_j} zlbc^{p,h}_j \\ &\forall j \in [1,n], p \in [1, maxp+1]: \\ &\quad x^p_j = \textstyle\sum_{h=1}^{nhubc_j} zubc^{p,h}_j\end{aligned} \tag{15}$$

This gives us the constraint:

$$\begin{aligned}&\forall p \in [1, maxp] : t_{p+1} - t_p \geq \\ &\quad \textstyle\sum_{j=1}^{n} \sum_{h=1}^{nhlbc_j} lbc^h_j \times zlbc^{p,h}_j \\ &\forall p \in [1, maxp] : t_{p+1} - t_p \leq \\ &\quad \textstyle\sum_{j=1}^{n} \sum_{h=1}^{nhubc_j} ubc^h_j \times zubc^{p,h}_j\end{aligned} \tag{16}$$

We must now express the condition for a certain pre-specified history of a job to match the schedule. Let $hmubc^{p,h}_j$ be a variable being one if it holds that the h:th pre-specified history used for an upper bound on execution time of task τ_j is matched in position p. Otherwise $hmubc^{p,h}_j$ is zero. Formally, we express this as:

$$\begin{aligned}&\forall j \in [1,n], p \in [1, maxp], h \in [1, nhubc_j]: \\ &\quad (hmubc^{p,h}_j = 1) \iff \\ &\qquad (lhubc^h_j \leq p-1) \wedge (\textstyle\bigwedge_{k=1}^{lhubc^h_j} (x^{p-k}_{ihubc^{h,k}_j} = 1))\end{aligned} \tag{17}$$

Observe that the variable $hmubc^{p,h}_j$ does not indicate that $zubc^{p,h}_j = 1$ – it only indicates that $zubc^{p,h}_j = 1$ is allowed. In order to get $zubc^{p,h}_j = 1$ we must have that this h is the pre-specified history with the smallest execution time among the ones that are matched. We say that such a pre-specified history is selected. Hence:

$$\begin{aligned}&\forall j \in [1,n], p \in [1, maxp], h \in [1, nhubc_j]: \\ &\quad (hmsubc^{p,h}_j = 1) \iff (hmubc^{p,h}_j = 1) \wedge \\ &\qquad (\forall h' \in [1, nhubc_j] : \\ &\qquad\quad ubc^{h'}_j < ubc^h_j \Rightarrow hmubc^{p,h'}_j = 0)\end{aligned} \tag{18}$$

and

$$\begin{aligned}&\forall j \in [1,n], p \in [1, maxp]: \\ &\quad \textstyle\sum_{h=1}^{nhubc_j} hmsubc^{p,h}_j \leq 1\end{aligned} \tag{19}$$

In addition, in order for $zubc^{p,h}_j = 1$ it must be that task τ_j is actually chosen to be executing by the dispatcher in position p. Hence:

$$\begin{aligned}&\forall j \in [1,n], p \in [1, maxp], h \in [1, nhubc_j]: \\ &\quad (zubc^{p,h}_j = 1) \iff (hmsubc^{p,h}_j = 1 \wedge x^p_j = 1)\end{aligned} \tag{20}$$

Note some constraints have forall-expressions. This enumeration is done before giving the constraints to an MILP solver so they pose no problems to us. A careful reader may wonder whether it can happen that there is an h' and h'' such that: $hmsubc^{p,h'}_j = 1$ and $hmsubc^{p,h''}_j = 1$ for the case that $ubc^{h'}_j = ubc^{h''}_j$. Note that (19) ensures that this cannot happen.

We use similar constraints to obtain a lower bound on the execution time of a job. Hence we have:

$$\begin{aligned}&\forall j \in [1,n], p \in [1, maxp], h \in [1, nhlbc_j]: \\ &\quad (hmlbc^{p,h}_j = 1) \iff \\ &\qquad (lhlbc^h_j \leq p-1) \wedge (\textstyle\bigwedge_{k=1}^{lhlbc^h_j} (x^{p-k}_{ihlbc^{h,k}_j} = 1))\end{aligned}$$

and

$$\begin{aligned}&\forall j \in [1,n], p \in [1, maxp], h \in [1, nhlbc_j]: \\ &\quad (hmslbc^{p,h}_j = 1) \iff (hmlbc^{p,h}_j = 1) \wedge \\ &\qquad (\forall h' \in [1, nhlbc_j] : \\ &\qquad\quad lbc^{h'}_j > lbc^h_j \Rightarrow hmlbc^{p,h'}_j = 0)\end{aligned}$$

and

$$\begin{aligned}&\forall j \in [1,n], p \in [1, maxp]: \\ &\quad \textstyle\sum_{h=1}^{nhlbc_j} hmslbc^{p,h}_j \leq 1\end{aligned} \tag{21}$$

$$\forall j \in [1,n], p \in [1, maxp], h \in [1, nhlbc_j]: \\ (zLBC^{p,h}_j = 1) \iff (hmslbc^{p,h}_j = 1 \wedge x^p_j = 1)$$

D. ftlastjob and requiring a single busy period

Let $busy_p$ be one if there is a job executing in position p; otherwise zero.

$$\forall p \in [1, maxp] : busy_p = \sum_{j=1}^{n} x^p_j$$

Let $lastbusy_0$ be one if for each position, it holds that there is no job executing in the position; otherwise zero. Let $lastbusy_p$, for $p \in [1, maxp]$ be one if there are jobs executing in all of the positions $1..p$ and there are no jobs executing in positions $p..maxp$; otherwise zero. We express $lastbusy_p$ and the requirement of a single busy period as:

$$\sum_{p=0}^{maxp} lastbusy_p = 1$$

and

$$\forall p \in [0, maxp] : \sum_{p'=1}^{p} busy_{p'} + \sum_{p'=p+1}^{maxp}(-busy_{p'}) - maxp \times lastbusy_p \geq -(maxp - p)$$

With these variables, *ftlastjob* is expressed as:

$$\forall p \in [0, maxp] : lastbusy_p = 1 \Rightarrow ftlastjob = t_{p+1}$$

All the variables are non-negative but this does not need to be stated explicitly.

E. Upper bound on real variables

Observe that feasibility and the objective function is dependent on the schedule in the positions [1,$maxp$] but it is not dependent on the schedule in position $maxp$+1. We can move the arrival times of jobs executing in position $maxp$+1 so that for those jobs, the inter-arrival times between two consecutive jobs of a task τ_j is exactly T_j and we still have a feasible solution with the same objective function. After this modification of the schedule, it holds that the last job finished at most

$$\begin{aligned} \text{SPAN} = \quad & \sum_{j=1}^{n}(\lceil \tfrac{t}{T_j} \rceil \times (\max_{h=1}^{nhistoriesUBC_j} ubc_j^h)) + \\ & 2 \times (\sum_{j=1}^{n}(\lceil \tfrac{t}{T_j} \rceil \times T_j)) \end{aligned} \tag{22}$$

time units after t_1. Therefore, we can, with no loss of feasibility and without affecting optimality, add the constraints:

$$\begin{aligned} & \forall p \in [1, maxp + 1] : \\ & \quad t_{maxp+1} - t_p \leq \text{SPAN} \end{aligned} \tag{23}$$

and

$$\begin{aligned} & \forall p \in [1, maxp + 1], j \in [1, n], k \in [1, njobs_j] : \\ & \quad A_{j,k} - t_p \leq \text{SPAN} \end{aligned} \tag{24}$$

and

$$\begin{aligned} & \forall p \in [1, maxp + 1], j \in [1, n], k \in [1, njobs_j] : \\ & \quad f_{j,k} - t_p \leq \text{SPAN} \end{aligned} \tag{25}$$

These constraints provide an upper bound on the difference between two variables that are real numbers. We will rely on this upper bound when translating, with a standard technique, logical constraints to MILP.

F. Feasibility and monotonicity

Note that there is a feasible solution. We can get a feasible solution as follows. Set:

$$\forall j \in [1, n], p \in [1, maxp] : x_j^p = 0$$

and

$$\forall j \in [1, n], k \in [1, njobs_j] : arrives_{j,k}^{maxp+1} = 1$$

This would give us that the objective function would be zero. This is clearly not optimal so an MILP solver would not return this but it ensures us that for each task set, there is a feasible solution. Let us consider t' and t'' where $0 < t' < t''$. We now prove that $requested(t') \leq requested(t'')$.

Consider the call $requested(t')$. Then we compute the number of jobs of each task using (3). Let $njobs'_j$ denote the value of $njobs_j$ for this call. Analogously, let $njobs''_j$ denote the value of $njobs_j$ for this call with the parameter t''. Clearly, we have $\forall j : njobs'_j \leq njobs''_j$. We know that the MILP in $requested(t')$ is feasible. Let us now consider the solution to this MILP, i.e., the assignment which maximizes the objective function. We can use its corresponding schedule (indicated by x_j^p values) as a feasible solution for the MILP of $requested(t'')$ and for each job which has not been assigned a position, set its arrival time to after t_{maxp+1}. This works because $\forall j : njobs'_j \leq njobs''_j$. Hence we have a feasible solution for the MILP of $requested(t'')$ with an objective function that is the same as the MILP of $requested(t')$. Hence, $requested(t')$ provides us with a lower bound on the objective function for the optimal solution of the MILP of $requested(t'')$. This gives us that: $requested(t') \leq requested(t'')$.

V. Computing $resp(i, q, t)$

We will now define the function $resp(i, q, t)$ as a solution to an optimization problem. The objective function is to maximize $f_{i,q} - A_{i,q}$ subject to the constraints we showed in the previous section and also subject to the constraints listed in this section below.

The finishing time is expressed as:

$$\forall p \in [1, maxp] : y_{i,q}^p = 1 \Rightarrow f_{i,q} = t_{p+1} \tag{26}$$

Job $\tau_{i,q}$ must execute in the busy period. Hence:

$$\sum_{p'=1}^{maxp} y_{i,q}^{p'} = 1 \tag{27}$$

VI. The new algorithm

Figure 5 shows our new algorithm.

Input: task set τ, task index i
Output: R_i

```
1: newL := min_{j=1}^{n}(min_{h=1}^{nhubc_j} UBC_j^h)
2: repeat
3:     oldL := newL
4:     < newL, seq > := requested(oldL)
5: until oldL ≥ newL
6: L := newL
7: Q_i = ⌈L/T_i⌉; maxRT := −1
8: for q = 1 to Q_i do
9:     < valid_{i,q}, R_{i,q}, seq > := resp(i, q, L)
10:    if valid_{i,q} and R_{i,q} > maxRT then
11:        maxRT := R_{i,q}
12: return maxRT
```

Figure 5. Algorithm to compute R_i.

VII. Computational Complexity

We will now prove the complexity of the schedulability analysis problem addressed in this paper by transforming a variant of Traveling Salesman (TS) to our problem[1].

TS-PATH-SD-ATLEASTCOST
Instance: An undirected graph G=(V,E) with a set of vertices V and a set of edges E and a source vertex $s \in V$ and a destination vertex $d \in V$. Each edge $e \in E$ has a non-negative cost which is an integer. K is a constant.
Question: Is there a path from s to d so that the path visits each of the other vertices exactly once and the cost along the path is at least K?

Lemma 1: TS-PATH-SD-ATLEASTCOST is NP-complete in the strong sense.

Proof: This can be proved as follows. HAMILTONIAN PATH BETWEEN TWO POINTS in [18, page 96] is known to be NP-complete. Let TS-PATH-SD-ATLEASTCOST-12 denote the problem TS-PATH-SD-ATLEASTCOST where costs of edges are either 1 or 2. Using restriction, one can show that TS-PATH-SD-ATLEASTCOST-12 is NP-complete in the strong sense. Since TS-PATH-SD-ATLEASTCOST-12 is a special case of TS-PATH-SD-ATLEASTCOST we obtain that TS-PATH-SD-ATLEASTCOST is NP-complete in the strong sense. ■

HBNS-UNSCHED-INT
Instance: A set of tasks and a priority assignment as described by the model in Section II but with T, D and LBC and UBC parameters being integers.
Question: Is there an assignment of arrival times to jobs such that for the resulting schedule it holds that there is a task with a job that misses its deadline?

[1]Discussions with (i) Sebastian Stiller and (ii) Nan Guan and Martin Stigge at the workshop RTSOPS 2011 provided us with the inspiration to use traveling salesman problem for reduction.

HBNS-UNSCHED-INT-R
Instance: A set of tasks and a priority assignment as described by the model in Section II but with T, D and LBC and UBC parameters being integers and the parameters fulfilling the following restrictions $T_1 = T_2 = \ldots = T_n = \infty$ and $D_1 = D_2 = \ldots = D_n$.
Question: Is there an assignment of arrival times to jobs such that for the resulting schedule it holds that there is a task with a job that misses its deadline?

Note that due to the restrictions in the instance of HBNS-UNSCHED-INT-R, answering the question of HBNS-UNSCHED-INT-R is equivalent to asking whether for the job set composed of one job of each task it is possible to order the jobs such that their cumulative execution time (considering history dependency) exceeds their common deadline.

Lemma 2: HBNS-UNSCHED-INT-R is NP-complete in the strong sense.

Proof: We will follow the standard method [18, page 30] of proving a problem NP-complete by reducing another problem to it. Using this methodology with TS-PATH-SD-ATLEASTCOST, it follows that we need to:

1) Show that HBNS-UNSCHED-INT-R is in NP.
2) Construct a transformation f from TS-PATH-SD-ATLEASTCOST to HBNS-UNSCHED-INT-R, and
3) Prove that f is a (polynomial) transformation.

Below, we discuss how to perform these three steps.

1) A solution to HBNS-UNSCHED-INT-R is a sequence of jobs. Given a solution to HBNS-UNSCHED-INT-R, we can compute the sum of the execution times in the sequence of jobs (taking into account the history-dependent execution times) and compare it to the common deadline. This computation has linear time-complexity as a function of the number of elements in the sequence and the number of histories. Therefore, a solution to HBNS-UNSCHED-INT-R can be verified in polynomial time.
2) We perform the mapping (named f) from a problem instance of TS-PATH-SD-ATLEASTCOST to a problem instance of HBNS-UNSCHED-INT-R as follows:
   a) For each vertex in the problem instance of TS-PATH-SD-ATLEASTCOST, except the source vertex s, create a corresponding task in the problem instance of HBNS-UNSCHED-INT-R.
   b) For each task τ_j in the problem instance of HBNS-UNSCHED-INT-R, set D_j to the bound K-1.
   c) For the destination vertex in TS-PATH-SD-

ATLEASTCOST, identify the corresponding task in HBNS-UNSCHED-INT-R and let this task have the lowest priority.

d) It can be seen that if there is a path p that is a solution to HBNS-UNSCHED-INT-R then this path can be used to create a sequence of the jobs from the tasks in HBNS-UNSCHED-INT-R. In the sequence, we say that the source vertex is the 0:th vertex in the sequence. We simply consider the k:th (where $k \geq 1$) vertex in the path. For this vertex, we identify the corresponding task. This job should be the k:th job in the sequence of jobs. In this way, we obtain a sequence of jobs from the path p. Analogously, it can be shown that if there is a sequence of jobs whose cumulative execution time exceeds the common deadline then a path that fulfills the corresponding instance of TS-PATH-SD-ATLEASTCOST can be obtained.

3) When performing the mapping (f) we iterate over the objects (vertices and edges) in the graph in the problem instance of TS-PATH-SD-ATLEASTCOST. Hence the time-complexity is $O(V + E)$. This is polynomial. Hence the mapping f has polynomial time-complexity.

■

Theorem 1: HBNS-UNSCHED-INT is NP-complete in the strong sense.

Proof: The theorem follows from the fact that HBNS-UNSCHED-INT-R is a restricted form of HBNS-UNSCHED-INT. ■

VIII. Conclusion

We have presented a model for describing non-preemptive fixed-priority scheduling where the execution time of a job depends on the actual schedule (sequence) of jobs executed before it. We have shown that exact schedulability analysis in this model is NP-complete in the strong sense and hence we should not expect to perform exact schedulability analysis with pseudo-polynomial time-complexity. For this reason, we presented an exact schedulability test which computes response times of tasks based on solving Mixed-Integer Linear Programs (MILP).

References

[1] B. Dougherty, J. White, R. Kegley, J. Preston, D. C. Schmidt, and A. Gokhale, "Optimizing DRE System Performance with the SMACK Cache Efficiency Metric," http://www.dre.vanderbilt.edu/~schmidt/PDF/RTSS-11.pdf.

[2] A. K. Mok and D. Chen, "A multiframe model for real-time tasks," in *Proc. of RTSS*, 1996.

[3] S. K. Baruah, D. Chen, S. Gorinsky, and A. K. Mok, "Generalized Multiframe Tasks," *Real-Time Systems (RTS)*, vol. 17, no. 1, 1999.

[4] S. K. Baruah, "A General Model for Recurring Real-Time Tasks," in *Proc. of RTSS*, 1998.

[5] N. T. Moyo, E. Nicollet, F. Lafaye, and C. Moy, "On Schedulability Analysis of Non-cyclic Generalized Multi-frame Tasks," in *Proc. of ECRTS*, 2010.

[6] S. K. Baruah, "The Non-cyclic Recurring Real-Time Task Model," in *Proc. of RTSS*, 2010.

[7] M. Stigge, P. Ekberg, N. Guan, and W. Yi, "The Digraph Real-Time Task Model," in *Proc. of RTAS*, 2011.

[8] A. Maxiaguine, S. Künzli, and L. Thiele, "Workload characterization model for tasks with variable execution demand," in *Proceedings of the Design, Automation and Test In Europe Conference and Exhibition (DAC '04)*, 2004.

[9] M. Jersak, R. Henia, and R. Ernst, "Context-aware performance analysis for efficient embedded system design," in *Proceedings of the Design, Automation and Test In Europe Conference and Exhibition (DAC '04)*, 2004.

[10] K. W. Tindell, "Adding time-offsets to schedulability analysis," University of York, Technical report YCS-221, 1994.

[11] G. Laurent, N. Rivierre, and M. Spuri, "Preemptive and Non-Preemptive Real-Time UniProcessor Scheduling," INRIA, Technical report RR-2966, 1996.

[12] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised," *Real-Time Systems (RTS)*, vol. 35, no. 3, 2007.

[13] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, 1973.

[14] M. Joseph and P. K. Pandya, "Finding Response Times in a Real-Time System," *The Computer Journal*, vol. 29, no. 5, 1986.

[15] J. P. Lehoczky, L. Sha, and Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," in *Proc. of RTSS*, 1989.

[16] J. P. Lehoczky, "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines," in *Proc. of RTSS*, 1990.

[17] M. G. Harbour, M. H. Klein, and J. P. Lehoczky, "Fixed priority scheduling periodic tasks with varying execution priority," in *Proc. of RTSS*, 1991.

[18] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.