Toward an Exact Simulation Interval for Multiprocessor Real-Time Systems Validation

Joumana Lagha, Jean-Luc Béchennec, Sébastien Faucou, Olivier-H Roux

Toward an Exact Simulation Interval for Multiprocessor Real-Time Systems Validation

Joumana Lagha, Jean-Luc Béchennec, Sébastien Faucou and Olivier-H Roux
Université de Nantes, École Centrale de Nantes, CNRS, LS2N, UMR 6004, F-44000 Nantes, France
Email: firstname.lastname@ls2n.fr

***Abstract*—In order to study the schedulability of complex real-time systems, simulation can be used. Of course, to achieve formal validation of schedulability, simulations must be run long enough such that the schedule repeats. An upper bound on the length of the simulation that is valid for a very wide class of systems running on top of identical multiprocessor platforms is given in a previous work. It is known that this bound is pessimistic. In this paper, we derive a characterization of the exact bound for the same class of systems and describe an algorithm for its computation. We use it to quantify the pessimism of the upper bound on a set of synthesized systems. We also give some directions to explore the complexity vs. tightness trade-off for this problem.**

Keywords–Real time scheduling; Multiprocessor; Simulation.

I. Introduction

The correctness of real-time software systems does not depend only on the value of results, but also on the date they are produced. A real-time software is usually composed of a set of recurring tasks that spawns jobs. Each job must be executed within a given deadline. Scheduling algorithms are used to allocate execution time to jobs. Schedulability analysis is used to validate that the resulting schedule meets all deadlines. For well-defined classes of systems, efficient schedulability tests exist [1]. For complex systems, that are not in one of these classes, it is sometimes possible to rely on simulation. More precisely, this is possible if the context does not yield scheduling anomalies, *ie.* when response times variations are monotonic with regards to other system parameters. In this paper, we will assume work under this hypothesis and refer the reader to Section 7 of [2] for a discussion on this point.

To achieve formal validation of the system, the simulation must be run on an interval long enough such that the schedule repeats. If the scheduler is deterministic and memoryless, then, if in this interval all jobs meet their deadline, it can be safely concluded that the system is schedulable. The length of this interval can be discovered during simulation by comparing each new state to those encountered so far. The main drawback of this approach is that it requires to memorize all states, so it quickly becomes intractable. An alternative consists of computing an upper bound B on the length of the simulation interval and then simulate the system on $[0, B)$. In 2016, Goossens *et al.* [2] proposed an upper bound that is valid for a wide class of systems: periodic asynchronous tasks with arbitrary deadlines and structural constraints (such as precedence, mutual exclusion and self-suspension) scheduled on top of an identical multiprocessor platform by any deterministic and memoryless algorithm.

It is known that this bound is pessimistic, especially because it does not take into account the processing power of the platform. Before looking for possible improvements, it is interesting to evaluate how pessimistic it is. To answer this question, we derive a characterization of the exact bound for the same class of systems and describe an algorithm for its computation. The algorithm relies on an enumeration of the state space and has factorial time complexity. On a set of synthetic systems, we find out that the pessimistic bound is at least twice too long when the number of tasks is greater than three times the number of processors. Based on the exact formulation of the bound, we also suggest directions to explore tightness vs. complexity trade-off for this problem.

The paper is organized as follows: in Section II, we review related works. In Section III, we define notations and expose the state-of-the-art. In Section IV, we give a characterization of the exact bound and derive an algorithm for its computation. In Section V, we compare the state-of-the-art and the exact bound on a set of synthetic benchmarks to quantify its pessimism. In Section VI, we present possible directions to explore the tightness vs. complexity trade-off before concluding the paper.

II. Related works

The first result on simulation intervals is obtained by Leung and Merrill [3] in 1980, with $O^{max} + 2H$ (where O^{max} is the maximum activation offset and H is the hyperperiod) as an upper bound for independent asynchronous task systems with constrained deadlines scheduled with a fixed-task priority algorithm. The same bound was later deemed valid for systems with arbitrary deadlines by Goossens and Devillers [4]. For multiprocessor platforms, Cucu and Goossens [5] derive in 2007 a result for independent asynchronous task systems (a task system is asynchronous if at least two tasks have their first activation on different dates) with arbitrary deadlines scheduled by a global fixed-task priority algorithm. They also prove that any feasible schedule generated by a deterministic and memoryless scheduler is ultimately periodic. In 2012, Baru *et al.* [6] proposed an upper bound on the simulation interval for asynchronous task systems with constrained deadlines subject to simple precedence constraints running on an identical multiprocessor platform and scheduled by any deterministic and memoryless algorithm. The same interval is used and tuned for fixed-job priority schedulers and independent tasks in Nélis *et al.* [7]. The most recent and general result is the one proposed by Goossens *et al.* [2] in 2016, that applies to a very large class of systems: asynchronous task systems with arbitrary deadlines, subject to structural constraints (precedence, mutual exclusion, self suspension), scheduled on an identical

multiprocessor platform by any deterministic and memoryless scheduler. This bound has a low complexity, is safe but not always tight. For many systems, it is very pessimistic and too big to be used for practical purpose. Thus, in this paper, we aim at deriving an exact bound. To do so, we relax the constraint on the complexity of the computation. Dues to its complexity, our bound can be computed for a restricted class of systems. For these systems, it provides an exact simulation interval. While deriving an exact bound, we also highlight how to explore the complexity vs. precision trade-off, paving the way to the development of low complexity yet precise bounds.

III. Upper bound on the simulation interval [2]

A. Model, notations, and definitions

$\mathbb{N}$ is the set of integer numbers. Let $\mathbf{v}$ be a vector of $\mathbb{N}^N$. $\forall i \in [1, N]$, $\mathbf{v}[i]$ is the ith element of the vector $\mathbf{v}$. We note that $\mathbf{0}$ the null vector: $\forall i \in [1, N]. \mathbf{0}[i] = 0$. The usual operators $+, -, \times, <$ and $=$ are used on vectors of $\mathbb{N}^N$ and are the pointwise extensions of their counterparts in $\mathbb{N}$. $\{x \mid P(x)\}$ is set set of all x such that predicate $P(x)$ is true. $[x \mid P(x)]$ is the list of all x such that predicate $P(x)$ is true.

Let $\Theta = \{\tau_1, \tau_2, \ldots, \tau_N\}$ be a set of N asynchronous periodic tasks, where each task τ_i is the 4-tuple of non negative integers $\langle O_i, C_i, T_i, D_i \rangle$, where O_i is the release time of the first job of τ_i, C_i is the execution time of τ_i, T_i is the period of τ_i, and D_i its deadline. We assume that periods and deadlines are unrelated (*i.e.*, D_i can be smaller than, equal to, or greater than T_i). $H = lcm_{\tau_i \in \Theta}\{T_i\}$ is the hyperperiod of Θ.

At runtime, each task τ_i spawns an infinite sequence of jobs $\tau_{i,1}, \tau_{i,2}, \ldots$. Job $\tau_{i,j}$ enters the system at date $a_{i,j} = O_i + (j-1)T_i$. It must be executed before date $d_{i,j} = a_{i,j} + D_i$.

Let $S(t)$ be the state of the system at date t. It is defined by $S(t) = (C_{rem_1}(t), \ldots, C_{rem_n}(t), \Omega_1(t), \ldots \Omega_n(t))$, where C_{rem_i} is the remaining work to process for the jobs of task τ_i activated prior to t, and $\Omega_i(t)$ is a decrementing clock counting the time until the next release of a job of τ_i.

Θ is executed on a platform composed of m identical processors. Jobs are scheduled by a deterministic and memoryless scheduler (see below). A given job is executed sequentially (no inner parallelism) but can migrate from one processor to another during its execution. It is assumed that there is no penalty to migrate from one processor to another. Moreover, it is assumed that a job cannot start its execution while all jobs of the same task activated before are not finished.

Definition 1 (Feasible schedule): A feasible schedule for Θ is an infinite schedule such that every job $\tau_{i,j}$ is fully executed in its time window $[a_{i,j}, d_{i,j}]$.

Definition 2 (Deterministic and memoryless scheduler): A scheduler such that the scheduling decision at time t is unique and depends only on the current state of the system.

Definition 3 (Valid simulation interval): Interval [0,B) is a valid simulation interval for Θ scheduled with a deterministic and memoryless scheduler if and only if $\exists (t_1, t_2) \in [0, B]^2$. $\quad t_1 \neq t_2 \wedge S(t_1) = S(t_2)$.

The model has two features that make its schedulability analysis complex: arbitrary deadlines and asynchronous activation. Both are sources of backlog between hyperperiods.

Definition 4 (Backlog): The backlog $\beta_i(t)$ of a task τ_i at date t is defined as the remaining work to be processed for jobs of τ_i activated strictly before t.

In the following, we assume that all hypotheses formulated in this section hold.

B. Ruling out asynchronous activations

To rule out the complexity arising from asynchronous task activation, Goossens *et al*. observe that a simple transformation can be applied to an asynchronous task set Θ to obtain a synchronous task set Θ' such that the length of the simulation interval of Θ' (considering any deterministic and memoryless scheduler) is not smaller than that of Θ. For each task $\tau_i = \langle O_i, T_i, D_i \rangle$, the transformation yields $\tau'_i = \langle 0, T_i, O_i + D_i \rangle$.

The idea is that all feasible schedules of Θ are also feasible schedules of Θ'. Thus, if a simulation is run for a duration long enough to validate any feasible schedule of Θ', it is also long enough to validate any feasible schedule of Θ. A detailed proof is given in [2].

Given this result, we can now reason as if we only had to handle synchronous task sets. Thus, we can now give a trivial upper bound on the backlog of a task at the end of a hyperperiod: $\forall q > 0. \beta_i(qH) \leq (O_i + D_i) - T_i$. From now on, we note $\beta_i^{max} = \max\{0, (O_i + D_i) - T_i\}$ the maximum backlog for task τ_i at any date $t = qH$ in any feasible schedule, and we note $\beta^{max} = \max_{\tau_i \in \Theta} \beta_i^{max}$.

C. Extension to structural constraints

The approach used to rule out asynchronous activation can be used to extend the result to systems with structural constraints. Structural constraints are defined as "*a relation between jobs or subjobs, forbidding some execution orders, preemptions, or insuring a minimal delay between the end of a job (or sub-job) and the start of another one*" [2]. Let Θ a system with structural constraints. Let Θ' denote the same system where all structural constraints have been removed. Obviously, all feasible schedules of Θ are also feasible schedules of Θ'. Thus, a valid simulation interval for Θ' is also a valid simulation interval for Θ.

D. Deriving the bound

In any non trivial synchronous system such that at least two tasks have different periods, the search for the upper bound of a valid simulation interval can be reduced to solutions of the form $B = qH$ (with q a positive integer) by definition of H (the hyperperiod of Θ) since local clocks are equal in $S(0)$ and $S(qH)$.

By definition, for any non negative integer q, $C_{rem_i}(qH) = \beta_i(qH)$, so in any feasible schedule, $C_{rem_i}(qH) \leq \beta_i^{max}$. Then, we can bound the number of different states of the system in any feasible schedule at the end of a hyperperiod: $|\{S(qH) \mid q \in \mathbb{N}\}| \leq \prod_{i \in [1,N]} (\beta_i^{max} + 1)$. Using the assumption that the scheduler is deterministic and memoryless, it is sufficient to run the simulation long enough to cover a number of hyperperiods equal to the number of different states at the end of a hyperperiod. If the schedule is not feasible then a deadline miss will be discovered. If the schedule is feasible,

either the same state will have been encountered twice, or all states will have been explored. This yields the bound:

$$B_0 = H \times \prod_{i \in [1,N]} (\beta_i^{max} + 1) \quad (1)$$

E. Non tightness of B_0

As claimed by the authors in [2], the bound is safe but not tight. This is illustrated in Figure 1. Let us consider a system with two tasks τ_1 and τ_2 such that $0 < \beta_1^{max} < \beta_2^{max}$, running on a monoprocessor platform. The size of the state

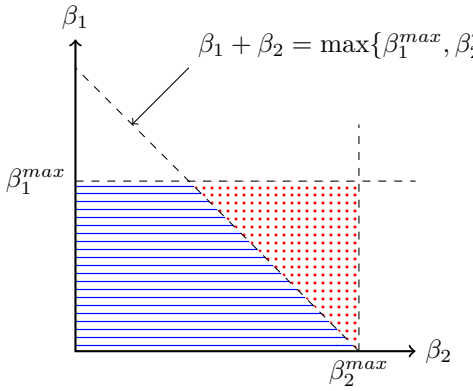

Figure 1. Illustration of the non-tightness of the bound: states in the red dotted area do not belong to any feasible schedule.

space considered by the bound B computed above is the number of points with integer coordinates in the rectangle of width β_2^{max} and height β_1^{max}. Now, let us consider the points in the red dotted area. They correspond to a pending work at the end of a hyperperiod, which is strictly greater than $\max\{\beta_1^{max}, \beta_2^{max}\} = \beta_2^{max}$. Starting from such a state at any $t = qH$, in any schedule, at least one job activated before t will finish after $t + \beta_2^{max}$ thus missing its deadline. We conclude that this state can not belong to any feasible schedule.

IV. Exact bound on the simulation interval

A. Characterization

We have seen with Figure 1 that B_0 fails to take into account diagonal constraints arising from the fact that the platform limits the execution parallelism and thus the maximum amount of cumulative backlog at the end of a hyperperiod. We can generalize this argument to derive a characterization of the bound as a set of linear constraints. Let $\Lambda \subseteq \Gamma$ be a subset of the task set. On a monoprocessor platform, the cumulative backlog at the end of a hyperperiod generated by tasks in Λ is bounded by $\max[\beta_i^{max} \mid \tau_i \in \Lambda]$ where $\max$ returns the maximum value of a list. On a 2-processor platform, execution parallelism allows us to achieve a higher bound: $\max_2[\beta_i^{max} \mid \tau_i \in \Lambda]$ where $\max_2$ returns the sum of the 2 greatest values of a list. Indeed, even if two jobs can be executed in parallelism, in a feasible schedule, they cannot overrun their deadlines. Thus, when the time is past the penultimate deadline, only one job among the jobs activated before the end of the hyperperiod, has not reached its deadline, so in a feasible schedule only this job could be running. Further generalizing this argument, on a m-processor platform, every $\Lambda \subseteq \Gamma$ yields the constraints $\sum_{\tau_i \in \Lambda} \beta_i \leq \max_m[\beta_i^{max} \mid \tau_i \in \Lambda]$ where $\max_m$ returns the sum of the m greatest values of a list. This allows us to characterize the number of possible states at the end of a hyperperiod (since we know that all local clocks are null at such instants, the state is truncated to its $C_{rem_i(t)}$ components).

$$\mathcal{S} = \Big\{ \mathbf{x} \mid \mathbf{x} \in \mathbb{N}^N \wedge \forall \Lambda \subseteq \Theta. \sum_{\tau_i \in \Lambda} \mathbf{x}[i] \leq \max_m[\beta_i^{max} \mid \tau_i \in \Lambda] \Big\} \quad (2)$$

From this, we can derive the exact value of the bound on the simulation interval:

$$B_1 = H \times |\mathcal{S}| \quad (3)$$

Note that using 2 to compute B involves computing the power set of Θ (to enumerate all possible values of Λ), which has $2^{|\Theta|}$ elements, and then enumerating the number of integer-coordinate points over a linear polyhedron defined by $2^{|\Theta|}$ constraints. It must also be noticed that B_0 corresponds to the enumeration of the points with integer coordinates of the smallest hyperrectangle that contains $\mathcal{S}$ and is exact when the definition of $\mathcal{S}$ involves no diagonal constraints, *i.e.*, when the number of tasks is not greater than the number of processors.

B. Computation of B_1

To count the number of states in $\mathcal{S}$, we rely on a fixed point computation. We start from state $\mathbf{0}$ and date qH. We expand the set of states time unit per time unit. Each time unit, we add states that have a cumulative backlog that fits in this extra time unit while taking into account platforms and tasks constraints. We stop once we have reached a fixed point over the set of states. We first describe the algorithm, then prove its termination, soundness, completeness, and apply it to a simple example.

1) One time unit mappings: Let us consider $\mathbf{Act} : \mathbb{N} \to \{0,1\}^N$ such that $\forall t \in [0, \beta^{max}). \forall i \in [1, N]. \mathbf{Act}(t)[i] = 0$ iff $t \leq \beta_i^{max}$, and $\mathbf{Act}(t)[i] = 1$ otherwise. That is to say $\mathbf{Act}(t)[i] = 1$ iff a job of τ_i activated before t has not necessarily reached its deadline at date t.

Let $Incr = \{\mathbf{v} \mid \mathbf{v} \in \{0,1\}^N \wedge \sum_{i=1}^{N} \mathbf{v}[i] \leq m\}$. An element of $Incr$ is a mapping of tasks to processors (remember that jobs of the same task must execute sequentially). As an example, for $N = 3$ tasks and $m = 2$ processors we have:

$$Incr = \left\{ \begin{bmatrix}0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\end{bmatrix}, \begin{bmatrix}0\\1\\0\end{bmatrix}, \begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}1\\1\\0\end{bmatrix}, \begin{bmatrix}0\\1\\1\end{bmatrix}, \begin{bmatrix}1\\0\\1\end{bmatrix} \right\}$$

Let $incr$ be a function from date to parts of $Incr$ such that $incr(t) = \{\mathbf{v} | \mathbf{v} \in Incr \wedge \mathbf{v} \times \mathbf{Act}(t) = \mathbf{v}\}$. $incr(t)$ describes the mappings of tasks to processors for $[t, t+1)$ in any feasible schedule, discarding those which execute a job of a task that has already missed its deadline. For example,

assuming $N = 3$ tasks, $m = 2$ processors, and $\mathbf{Act}(1) = \begin{bmatrix}0\\0\\1\end{bmatrix}$, then we have:

$$incr(1) = \left\{ \begin{bmatrix}0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\end{bmatrix} \right\}$$

Lemma 1: A one time unit mapping of tasks onto processor $\mathbf{inc}$ for interval $[qH + t, qH + t + 1)$ for any non negative integer q is part of a feasible schedule if and only if $\mathbf{inc} \in incr(t)$.

Proof: Follows from the definition of $incr$. ■

Definition 5 (Possible mapping): A possible mapping is a one time unit mapping of tasks onto processor $\mathbf{inc} \in incr(t)$.

2) Fixed point algorithm: Let S be a set of states. We define the successor of S by elapsing one time unit from date $qH + t$ to $qH + t + 1$ as follows:

$$next(S, t) = \{\mathbf{s} + \mathbf{inc} \mid \mathbf{s} \in S \wedge \mathbf{inc} \in incr(t)\} \quad (4)$$

Given this definition of $next$, the set of possible states of the system at the end of a hyperperiod in any feasible schedule is the smallest fixed point of:

$$\begin{cases} S_0 = \{\mathbf{0}\} \\ S_{n+1} = S_n \cup next(S_n, n) \end{cases} \quad (5)$$

The resulting bound B_1 can be computed with algorithm in Figure 2 below.

Algorithm 1 Computation of B_1

Require: Θ, m
Ensure: B_1
  $S \leftarrow \mathbf{0}$
  $t \leftarrow 0$
  while $next(S, t) \not\subseteq S$ **do**
    $S \leftarrow S \cup next(S, t)$
    $t \leftarrow t + 1$
  end while
  $B_1 = H \times |S|$
  return B_1

Figure 2. Fixed point algorithm for the computation of the exact bound B_1.

3) Termination: Recall that $\beta^{max} = \max_{\tau_i \in \Theta}\{\beta_i\}$ is the greatest possible backlog of any task at the end of a hyperperiod. From the definition of function $incr$, we have $incr(\beta^{max}) = \{\mathbf{0}\}$ and then the smallest fixed point is met at worst in β^{max} steps. During the computation of B_1 each state of $\mathcal{S}$ has to be stored. The number of states is upper bounded by the B_0. During the computation of B_1, each new state has to be compared to the set of states already explored. Hence our algorithm has also a factorial complexity in the state space size. A more detailed analysis, including a complexity analysis of the problem is out of the scope of this paper.

4) Soundness and Completeness :

Theorem 1 (Completeness and Soundness): $\mathbf{s} \in \mathcal{S}$ if and only if $\mathbf{s}$ is reachable by a feasible schedule from $\mathbf{0}$.

Proof: Soundness. *Ab absurdo*. Assume that there exists a state $\mathbf{s} \in \mathcal{S}$ which is not reachable by a feasible schedule from $\mathbf{0}$. Then, there exists $t \in [0, \beta^{max})$, a state $\mathbf{s}_t \in S_t$ that is reachable through possible mappings from $\mathbf{0}$ and a state $\mathbf{s}_{t+1} \in S_{t+1}$ that is not reachable through possible mappings from $\mathbf{0}$ such that $\mathbf{s}_{t+1} \in next(\{\mathbf{s}_t\}, t)$. Then, there exists $\mathbf{inc} \in incr(t)$ such that $\mathbf{s}_{t+1} = \mathbf{s}_t + \mathbf{inc}$ whereas $\mathbf{inc}$ is not possible at date t contradicting Lemma 1.

Completeness. *Ab absurdo*. Assume that there exists a state $\mathbf{s}$ which is reachable through possible mappings from $\mathbf{0}$ and such that $\mathbf{s} \notin \mathcal{S}$. Then, there exists $t \in [0, \beta^{max})$ and a state $\mathbf{s}_{t+1}$ that is reachable by a possible mapping from $\mathbf{s}_t \in S_t$ such that $\mathbf{s}_{t+1} \notin next(\{\mathbf{s}_t\}, t)$. Then, there exists $\mathbf{inc}$ such that $\mathbf{s}_{k+1} = \mathbf{s}_k + \mathbf{inc}$ and $\mathbf{inc} \notin incr(t)$ whereas $\mathbf{inc}$ is possible at date t contradicting Lemma 1. ■

5) Example: Consider a system with $N = 3$ tasks running on a platform with $m = 2$ processors. The charactetistics of the tasks are such that $\begin{matrix}\beta_1\\\beta_2\\\beta_3\end{matrix} = \begin{bmatrix}1\\1\\3\end{bmatrix}$. The possible mappings of tasks to processors in the first time unit after a hyperperiod is given by:

$$incr(0) = Incr = \left\{ \begin{bmatrix}0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\end{bmatrix}, \begin{bmatrix}0\\1\\0\end{bmatrix}, \begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}1\\1\\0\end{bmatrix}, \begin{bmatrix}0\\1\\1\end{bmatrix}, \begin{bmatrix}1\\0\\1\end{bmatrix} \right\}$$

and possible mappings in the following time units are given by:

$$incr(1) = incr(2) = \left\{ \begin{bmatrix}0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\end{bmatrix} \right\} \text{ and } incr(3) = \left\{ \begin{bmatrix}0\\0\\0\end{bmatrix} \right\}$$

Now, let us compute the smallest fixed point.

$$S_0 = \left\{ \begin{bmatrix}0\\0\\0\end{bmatrix} \right\}, \quad S_1 = \left\{ \begin{bmatrix}0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\end{bmatrix}, \begin{bmatrix}0\\1\\0\end{bmatrix}, \begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}1\\1\\0\end{bmatrix}, \begin{bmatrix}0\\1\\1\end{bmatrix}, \begin{bmatrix}1\\0\\1\end{bmatrix} \right\},$$

$$S_2 = \left\{ \begin{bmatrix}0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\end{bmatrix}, \begin{bmatrix}0\\1\\0\end{bmatrix}, \begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}1\\1\\0\end{bmatrix}, \begin{bmatrix}0\\1\\1\end{bmatrix}, \begin{bmatrix}1\\0\\1\end{bmatrix}, \begin{bmatrix}0\\0\\2\end{bmatrix}, \begin{bmatrix}1\\1\\1\end{bmatrix}, \begin{bmatrix}0\\1\\2\end{bmatrix}, \begin{bmatrix}1\\0\\2\end{bmatrix} \right\}$$

and then

$$\mathcal{S} = S_3 = \left\{ \begin{bmatrix}0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\end{bmatrix}, \begin{bmatrix}0\\1\\0\end{bmatrix}, \begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}1\\1\\0\end{bmatrix}, \begin{bmatrix}0\\1\\1\end{bmatrix}, \begin{bmatrix}1\\0\\1\end{bmatrix}, \begin{bmatrix}0\\0\\2\end{bmatrix}, \begin{bmatrix}1\\1\\1\end{bmatrix}, \begin{bmatrix}0\\1\\2\end{bmatrix}, \begin{bmatrix}1\\0\\2\end{bmatrix}, \begin{bmatrix}0\\0\\3\end{bmatrix}, \begin{bmatrix}1\\1\\2\end{bmatrix}, \begin{bmatrix}0\\1\\3\end{bmatrix}, \begin{bmatrix}1\\0\\3\end{bmatrix} \right\}$$

We obtain $B_1 = H \times |\mathcal{S}| = 15H$. With the same system, we have $B_0 = H \times (2 \times 2 \times 4) = 16H$. The state that was discarded in B_1 is $\begin{bmatrix}1\\1\\3\end{bmatrix}$ because it requires 5 time units of computation but a valid schedule cannot have more than $\max_2\{1, 1, 3\} = 4$ time units of pending work.

V. Experimentation

A. Setup

The computation of B_1 has factorial time complexity so it does not scale to big systems. Its main interest is to provide a reference to assess the tightness of approximate bounds. In particular, it is worth asking when B_0 is a reasonable

TABLE I. DETAILS AND PARAMETERS FOR THE 5 SERIES OF EXPERIMENTS

Series	N	m	β^{max}	Watchdog expiry time	Timeout (%)
1	$1 \leq N \leq 12$	$1 \leq m \leq 8$	20	15 min for $N \leq 9$ and 20 min for $N > 9$	18.47
2	$1 \leq N \leq 12$, 11 excluded	$1 \leq m \leq 8$	10	15 min for $N \leq 10$ and 45 min when N is 12	8.92
3	$1 \leq N \leq 12$	$1 \leq m \leq 8$	5	15 min	7
4	$k \leq N \leq 9$ with $k = 2 \times m$	$1 \leq m \leq 4$	$(10 - N) \times 8$	10 min	0.75
5	16	4	$2 \leq \beta^{max} \leq 6$	10 min	29

approximation, and if it is worth searching for less pessimistic approximations for certain systems. Thus, in this section, we evaluate the pessimism of bound B_0 with respect to B_1. We also provide some results concerning the resource consumptions of the computation of B_1 to characterize the range of systems that it can solve.

We implemented algorithm 2 in C, using red-black trees as the data structure for state sets. Computations have been run on a Debian GNU/Linux 8.8 system (kernel 3.16.0-4) with Intel(R) Xeon(R) CPU E5-2620 @ 2.00GHz and 128GB RAM. The evaluation set is based on five series of experiments. In each case, we take 20 samples per point (a point is defined by a number of tasks, a number of processors, and a value for β^{max}), and the algorithm is applied to each point. For each sample, the maximum backlog of each task is randomly generated between 1 and β^{max} using a uniform distribution. The code is instrumented to report execution time and maximum memory consumption of the computation of B_1. Lastly, a watchdog is used to stop the computation of B_1 after a pre-defined amount of time. Parameter values for each series are shown in Table I.

We provide a set of graphs that have been chosen to be as representative as possible of the data set. For each point, we represent the arithmetic average as well as minimum and maximum values among all 20 samples. In each figure the *y-axis* represents the ratio B_1/B_0 as a percentage. The quantity associated with the *x-axis* varies so it is specified in each figure.

B. Pessimism of B_0

Figure 3 shows the result when the number of tasks increases for a given number of processors. Figure 4 shows the result when the number of processors increases for a given number of tasks. As expected, both figures show that when the number of diagonal constraints increases, B_0 becomes more pessimistic. From 2, diagonal constraints appear for sets $\Lambda \subseteq \Theta$ such that $|\Lambda| > m$, *i.e.*, when the platform does not offer enough parallelism. Thus, the number of diagonal constraints increases with $\frac{N}{m}$. Figure 5 plots $\frac{B_1}{B_0}$ against $\frac{N}{m}$. It shows that, on this data series, B_0 quickly becomes a loose approximation of B_1: when $\frac{N}{m}$ becomes greater than 3, $\frac{B_1}{B_0}$ falls to 50 %, and below for higher values of $\frac{N}{m}$. The complexity of the computation of B_1 does not allow us to extend the plot further but it is expected that, as the number of linear constraint increases, $\frac{B_1}{B_0}$ asymptotically tends to zero.

Figure 7 plots $\frac{B_1}{B_0}$ against the standard deviation computed over the list $[\beta_i^{max} \mid \tau_i \in \Theta]$. Although it is not as clear as the impact of $\frac{N}{m}$, it shows that when the standard deviation is small, B_0 tends to be more pessimistic. Figure 7 also shows that similar values of $\frac{B_1}{B_0}$ can be reached for different values of β^{max} with similar dispersion of values of β_i^{max}. An

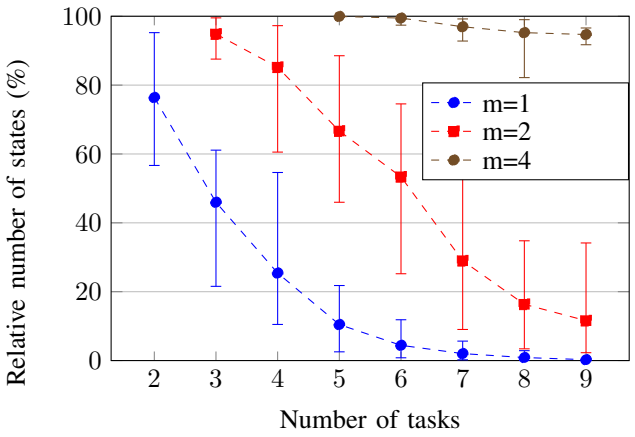

Figure 3. $N = 1$ to 9 tasks, $m = 1$ to 4 processors, $\beta^{max} = 20$.

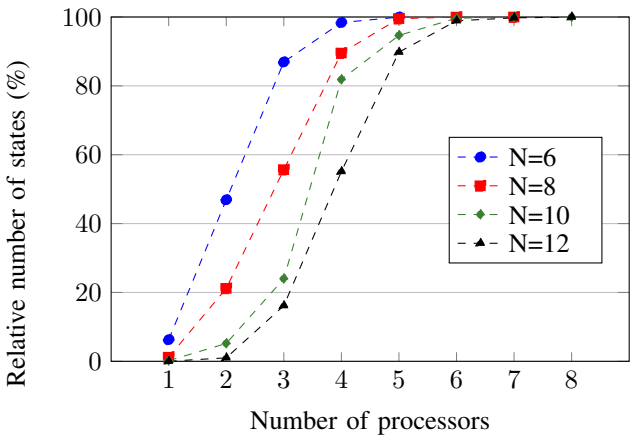

Figure 4. $N \in \{6, 8, 10, 12\}$ tasks, $m = 1$ to 8 processors, $\beta^{max} = 10$.

intuitive interpretation can be formulated from the example of figure 1: if $\beta_1^{max} = \beta_2^{max}$ then the diagonal constraints $\beta_1 + \beta_2 \leq \max\{beta_1^{max}, \beta_2^{max}\}$ removes half of the points of B_0 and this is the worst case. So, the closer the values of β_i^{max} in numerous mismatch, the more states it removes. And of course, a small standard deviation denotes a system with a small dispersion of β_i^{max} values.

C. Scalability of algorithm 2

The computation of B_1 requires a factorial number of comparisons with regards to the size of the state space of the system. Thus, it is sensible to every parameter that has an impact on the state space: number of tasks N, of processors m, and the maximum backlogs of tasks β_i^{max}.

Table II groups results for $N = 16$ and $m = 4$. In this case, the average execution time increases from 6.25 s to 385 s

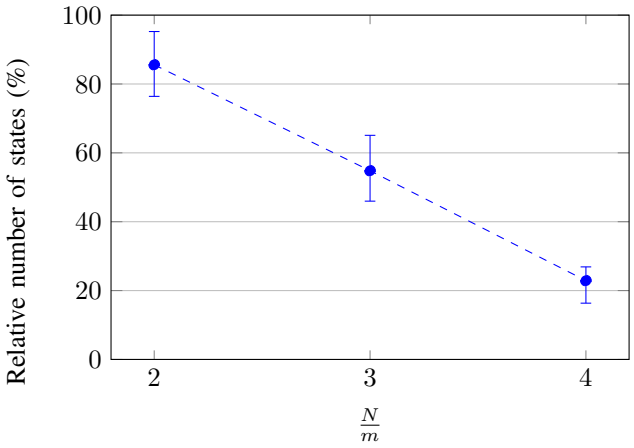

Figure 5. $N = 1$ to 12 tasks, $m = 1$ to 4 processors, $\beta^{max} = 20$.

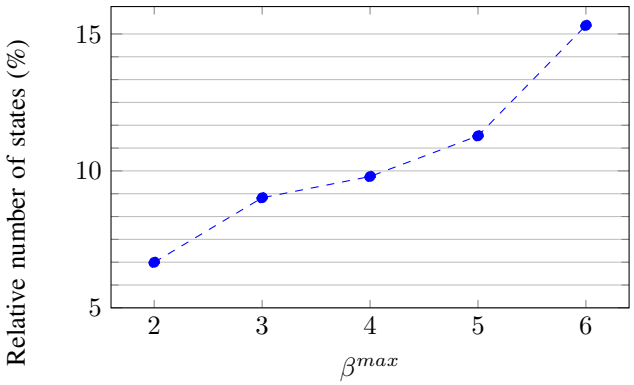

Figure 6. $N = 16$ tasks, $m = 4$ processors, $\beta^{max} = 2$ to 6.

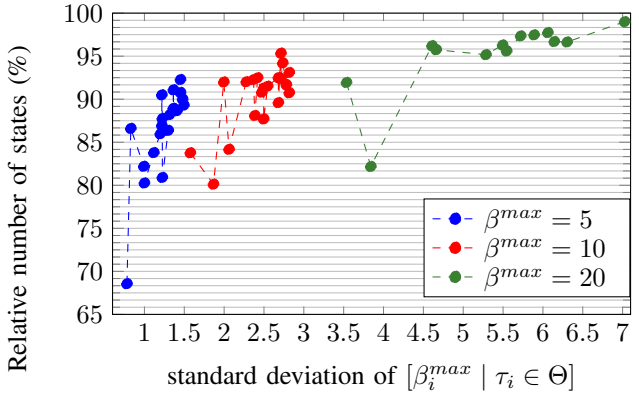

Figure 7. $N = 8$ tasks, $m = 4$ processors, $\beta^{max} \in \{5, 10, 20\}$.

TABLE II. EXECUTION TIME (IN SECONDS), WHEN $N = 16$ AND $m = 4$

β^{max}	Average	Maximum	Minimum	Standard deviation
3	6.25	22	microseconds	6.146244039
4	124.68	364	2	143.8439944
5	385	599	94	169.7838733

TABLE III. EXECUTION TIME (IN SECONDS), WHEN $m = 4$ AND $\beta^{max} = 20$

N	Average	Maximum	Minimum	Standard deviation
6	5.95	33	microseconds	9.827324956
7	124.2	511	1	136.6607786
8	254.54	701	18	302.4903777

just by increasing β^{max} from 3 to 5. Table III groups results for $m = 4$ and $\beta^{max} = 20$. In this case, the average execution time increases from 5.95 s to 254.54 s just by increasing N from 6 to 8. Lastly, Table IV groups results for $N = 8$ and $\beta^{max} = 20$. In this case, the average time increases from 1.05 s to 130.4 s just by increasing the m from 1 to 3. From these three tables, the influence of the individual β_i^{max} values on the overall execution time can also be seen: in Table II for example, with $N = 16$, $m = 4$ and $\beta^{max} = 5$, the execution time varies from 94 s to 599 s.

Similar results are observed for memory occupation. Indeed, the whole state space has to be stored. Table V shows for instance the maximum memory occupation for varying values of N and m when $\beta^{max} = 20$. As expected, the time and space complexity of algorithm 2 makes it impossible to deal with systems that have too large a state space. Nevertheless, many industrial systems use small multicore platforms. For instance, the 32 bit Microcontroller TriCore family developed by Infineon for the embedded automotive market offers platforms with 1 to 6 cores. Moreover, not all tasks in these systems have a non null backlog at hyperperiod boundaries, so B_1 could be of practical use for these systems. Additional experiments on industrial benchmarks are required to provide an answer to this question and it is out of the scope of this paper.

VI. Conclusion

The problem addressed in this paper is to compute an exact bound on the simulation interval for systems of asynchronous periodic tasks with arbitrary deadlines subject to structural

TABLE IV. EXECUTION TIME (IN SECONDS), WHEN $N = 8$ AND $\beta^{max} = 20$

m	Average	Maximum	Minimum	Standard deviation
1	1.05	6	microseconds	1.637552731
2	98	343	4	95.8200067
3	130.4	899	1	361.2071865

TABLE V. RESIDENT SET SIZE USED (IN MB), WHEN $\beta^{max} = 20$

N	m	Average	Maximum	Minimum	Std dev.
5	1	4.056	5.492	3.980	0.3380934782
6	1	13.935	47.980	3.988	13.92257703
7	1	108.555	640.840	3.812	166.1081405
8	1	1398.797	8286.392	66.756	2350.883606
9	1	15832.102	103894.004	640.600	28790.4245
5	2	8.250	19.372	3.980	5.43988676
6	2	35.805	168.132	3.988	39.78900124
7	2	238.146	940.680	10.796	229.4609528
8	2	3027.923	10032.492	147.412	3030.038648
9	2	27974.981	95923.060	1027.404	15778.69099

constraints scheduled by any deterministic and memoryless algorithm on a uniform multiprocessor platform. A very simple yet pessimistic solution for this problem is already known in the state-of-the-art. We formulate a characterization of the bound that involves the cardinal of the set of points with integer coordinates in a polyhedron defined by an exponential number of linear constraints. We propose and prove a fixed point algorithm to compute this set, which has factorial time complexity.

We rely on an implementation of this algorithm to estimate the pessimism of the bound known from the state-of-the-art through a set of experiments on synthetic systems. In the results of these experiments we observe two points: (i) the bound from the state-of-the-art quickly becomes a loose estimation of the exact bound when the number of tasks becomes greater than the number of processors of the platform; (ii) the time complexity of our algorithm is too high to deal with anything but small systems. From these two points, we conclude that there is an interest in looking at approximate bounds that lie in the middle between the state-of-the-art and the exact bound. Our formulation of the problem as a linear system already gives us a direction. The state-of-the-art provides a simple but pessimistic solution by discarding all diagonal constraints, while the exact bound does the opposite. So, as a direct follow-up to the work described here, we will now explore the idea to take into account a subset of the diagonal constraints to find a good trade-off between precision and time complexity.

REFERENCES

[1] R. I. Davis, A. Zabos, and A. Burns, "Efficient exact schedulability tests for fixed priority real-time systems," IEEE Transactions on Computers, vol. 57, no. 9, 2008, pp. 1261–1276.

[2] J. Goossens, E. Grolleau, and L. Cucu-Grosjean, "Periodicity of real-time schedules for dependent periodic tasks on identical multiprocessor platforms," Real-Time Syst, vol. 52, no. 6, 2016, pp. 808–832.

[3] J. Leung and J. Merrill, "A note on preemptive scheduling of periodic, real-time tasks," Information Processing Letters, vol. 11, no. 3, 1980, p. 115–118.

[4] J. Goossens and R. Devillers, "Feasibility intervals for the deadline driven scheduler with arbitrary deadlines," in Proceedings Sixth International Conference on Real-Time Computing Systems and Applications. RTCSA'99 (Cat. No.PR00306), 1999, pp. 54–61.

[5] L. Cucu and J. Goossens, "Feasibility intervals for multiprocessor fixed-priority scheduling of arbitrary deadline periodic systems," in 2007 Design, Automation Test in Europe Conference Exhibition, 2007, pp. 1–6.

[6] J. Baro, F. Boniol, M. Cordovilla, E. Noulard, and C. Pagetti, "Off-line (optimal) multiprocessor scheduling of dependent periodic tasks," in Proceedings of the 27th annual ACM symposium on applied computing (SAC), 2012, pp. 1815–1820.

[7] V. Nélis, P. Yomsi, and J. Goossens, "Feasibility intervals for homogeneous multicores, asynchronous periodic tasks, and fjp schedulers," in Proceedings of the 21st international conference on real-time networks and systems, 2013, pp. 277–286.