Toward an exact simulation interval for multiprocessor real-time systems validation

Presenter: Joumana LAGHA, third year PHD student at École centrale de Nantes.
email address: joumana.lagha@ls2n.fr.
Authors: Joumana LAGHA,
Jean-Luc BÉCHENNEC,
Sébastien FAUCOU,
and Olivier-H ROUX

Resume

Joumana LAGHA

- Third year PHD student in Computer Science at **École centrale de Nantes**.
- Member of real time system team at **LS2N laboratory**.
- Master's Degree in Automation, Robotics and Applied Computer Science, specializing in real-time control and supervision from **École centrale de Nantes**.
- Engineering degree in electrical and electronic engineering specializing in industrial computing and control from **Lebanese university**.

Research interest of real time system team of LS2N

Bringing together researchers from the University of Nantes, the École Centrale de Nantes and the CNRS, the STR team focuses on the entire development cycle of real-time systems, and mainly on their software aspect. It thus develops original work combining formal methods, operational reliability, execution platforms and real-time scheduling. Most of these works are implemented in practice in software freely available on `http://www.rts-software.org`.
Team topics :

- Execution supports
- Scheduling
- Formal methods

Table of contents

Table of contents

Real-time computing refers to applications that :

- Compute correct results.
- Perform on-time : their goal is not to be fast, but rather to be on-time.

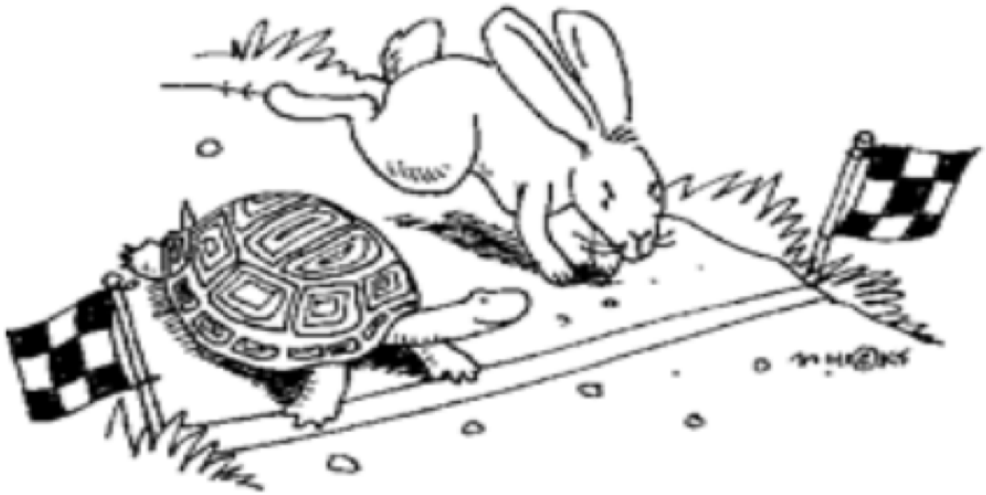

Figure – Real-time is not real-fast

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. Each task τ_i is the 4-tuple of non negative integers $\langle O_i, C_i, T_i, D_i \rangle$:

- O_i : the release time of the first job of τ_i.
- C_i : the execution time of τ_i.
- T_i : the period of τ_i.
- D_i : the deadline of τ_i, periods and deadlines are unrelated (*i.e.*, D_i can be smaller than, equal to, or greater than T_i).

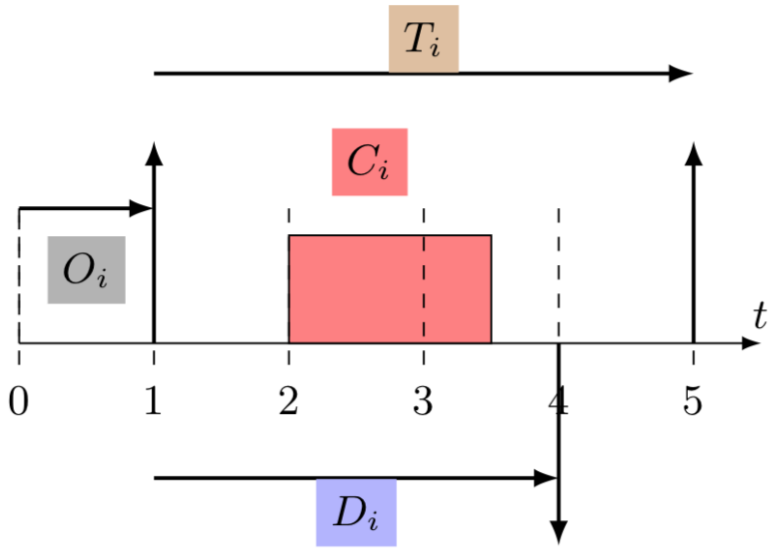

Figure – Task's characteristics

Scheduling algorithms are used to allocate execution time to jobs. Schedulability analysis is used to validate that the resulting schedule meets all deadlines.
In order to test the schedulability of the system, two different methods are known :

- Schedulability tests for well defined classes of systems[1].
- Simulation approach for complex systems that are not in one of previous classes.

Condition for the simulation approach :

To test the schedulability of the system Simulation approaches can be adopted if the context does not yield scheduling anomalies, *ie.* when response times variations are monotonic with regards to other system parameters [2].

Condition for the simulation approach :

To test the schedulability of the system Simulation approaches can be adopted if the context does not yield scheduling anomalies, *ie.* when response times variations are monotonic with regards to other system parameters [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.

Condition for the simulation approach :

To test the schedulability of the system Simulation approaches can be adopted if the context does not yield scheduling anomalies, *ie.* when response times variations are monotonic with regards to other system parameters [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.

Simulation approach principle :

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.

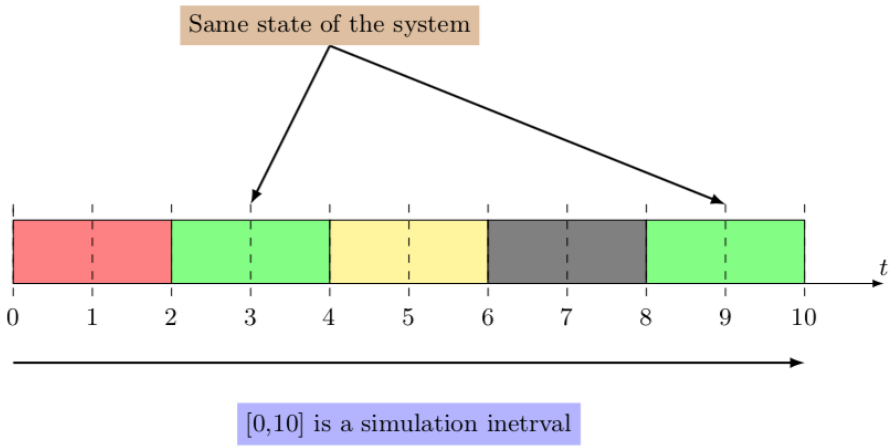

Figure – Evolution of system state

Drawback of the simulation interval approach :

The main drawback of this approach is that it requires to memorize all states, so it quickly becomes intractable.

Solution :

An alternative consists of computing an upper bound B on the length of the simulation interval and then simulate the system on [0,B).

Table of contents

- The first result on simulation intervals is obtained by Leung and Merrill [3] in 1980 for independent asynchronous task systems with constrained deadlines scheduled with a fixed-task priority algorithm.

- The same bound was later deemed valid for systems by Goossens and Devillers [4] extended the previous result to tasks with arbitrary deadlines.

- 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.

- 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 previous 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.

Table of contents

- Let $\Theta = \{\tau_1, \tau_2, \ldots, \tau_N\}$ be a set of N asynchronous periodic tasks (a task system is asynchronous if at least two tasks have their first activation on different dates).
- Each task τ_i is the 4-tuple of non negative integers $\langle O_i, C_i, T_i, D_i \rangle$.
- $H = lcm_{\tau_i \in \Theta}\{T_i\}$ 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 $d_{i,j} = a_{i,j} + D_i$.

Conditions and assumptions :

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

State of the system :

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.
- $\Omega_i(t)$ is a decrementing clock counting the time until the next release of a job of τ_i.

Definitions :

- **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 $[a_{i,j}, d_{i,j}]$.

Definitions :

- **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 $[a_{i,j}, d_{i,j}]$.
- **Valid simulation interval :**
  Interval [0,B) is a valid simulation interval for Θ scheduled with a deterministic and memoryless scheduler if and only if $\exists(t_1, t_2) \in [0, B]^2. \quad t_1 \neq t_2 \wedge S(t_1) = S(t_2)$.

Definitions :

- **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 $[a_{i,j}, d_{i,j}]$.
- **Valid simulation interval :**
  Interval [0,B) is a valid simulation interval for Θ scheduled with a deterministic and memoryless scheduler if and only if $\exists (t_1, t_2) \in [0, B]^2. \quad t_1 \neq t_2 \wedge S(t_1) = S(t_2)$.
- **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.

Problem :

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

Solution :

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.

The transformation is as follows :
For each task $\tau_i = \langle O_i, T_i, D_i \rangle$, the transformation yields $\tau_i' = \langle 0, T_i, O_i + D_i \rangle$.

Consequence :

we can now reason as if we only had to handle synchronous task sets.

Notations :

- $\beta_i^{max} = \max\{0, (O_i + D_i) - T_i\}$ the maximum backlog for task τ_i at any date $t = qH$ in any feasible schedule.
- $\beta^{max} = \max_{\tau_i \in \Theta} \beta_i^{max}$.

Principle :

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 $B = qH$ 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, $C_{rem_i}(qH) \leq \beta_i^{max}$.

Results :

We can bound the number of different states of the system in any feasible schedule at the end of an hyperperiod : $|\{S(qH) \mid q \in N\}| \leq \prod_{i \in [1,N]} (\beta_i^{max} + 1)$. Hence, 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.

This yields the bound :

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

As claimed by the authors in[2], the bound is safe but not tight.
This is illustrated in following figure. 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.

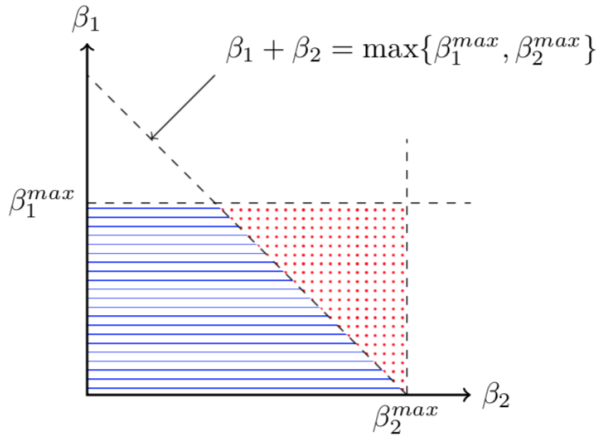

Figure – Illustration of the non-tightness of the bound : states in the red dotted area do not belong to any feasible schedule.

The size of the state 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.

Table of contents

In our work, we derive a characterization of the exact bound for the same class of systems and we describe an algorithm for its computation. We adopt all the previous definitions and assumptions.

By taking into account the diagonal constraints arising from the fact that the platform limits the execution parallelism we can 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 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.

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

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

$$B_1 = H \times |\mathcal{S}| \tag{3}$$

We must notice that :

- using equation(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.

We must notice that :

- using equation(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.
- 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.

To count the number of states in $\mathcal{S}$, we rely on a fixed point computation.

1. We start from state $\mathbf{0}$ and date qH.
2. We expand the set of states time unit per time unit.
3. 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.
4. We stop once we have reached a fixed point over the set of states.

Table of contents

- 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 approximation, and if it is worth searching for less pessimistic approximations for certain systems.
- 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 our fixed point algorithm 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.

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

Figure – DETAILS AND PARAMETERS FOR THE 5 SERIES OF EXPERIMENTS

- 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.

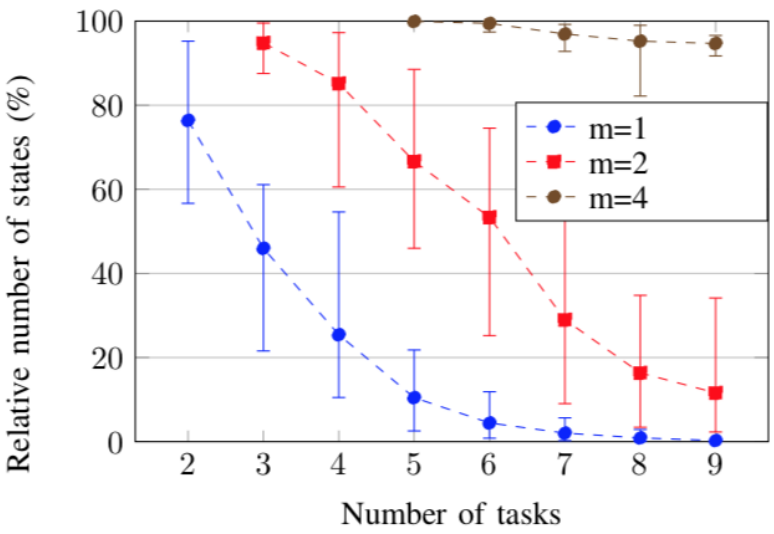

Figure – $N = 1$ to 9 tasks, $m = 1$ to 4 processors, $\beta^{max} = 20$.

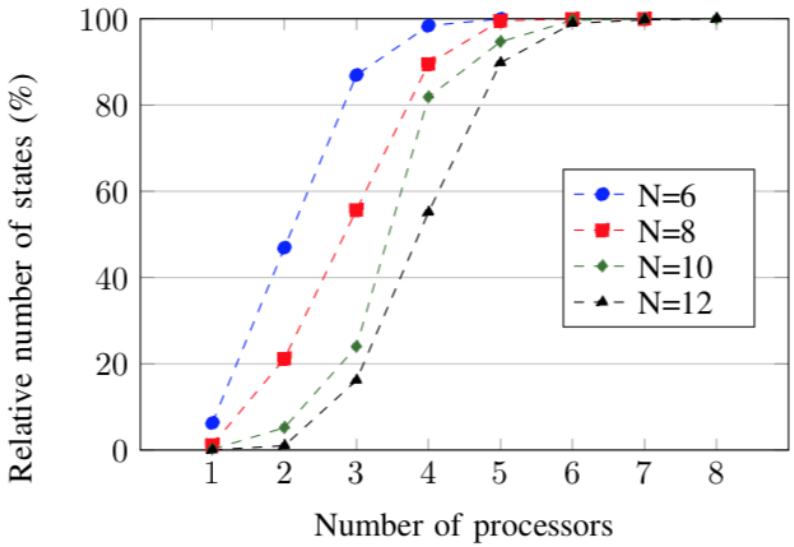

Figure – $N \in \{6, 8, 10, 12\}$ tasks, $m = 1$ to 8 processors, $\beta^{max} = 10$.

Above figures show the result when the number of tasks increases for a given number of processors and 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.

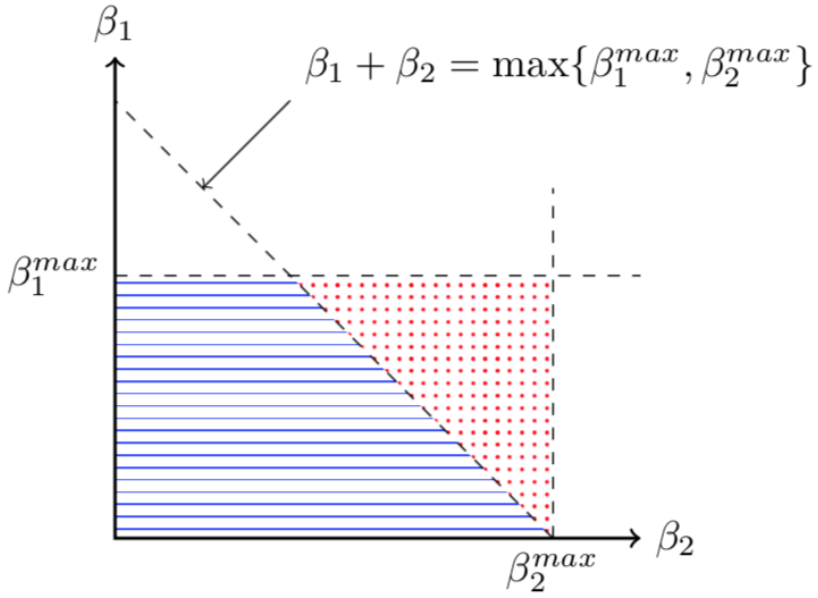

Figure – Illustration of the non-tightness of the bound : states in the red dotted area do not belong to any feasible schedule.

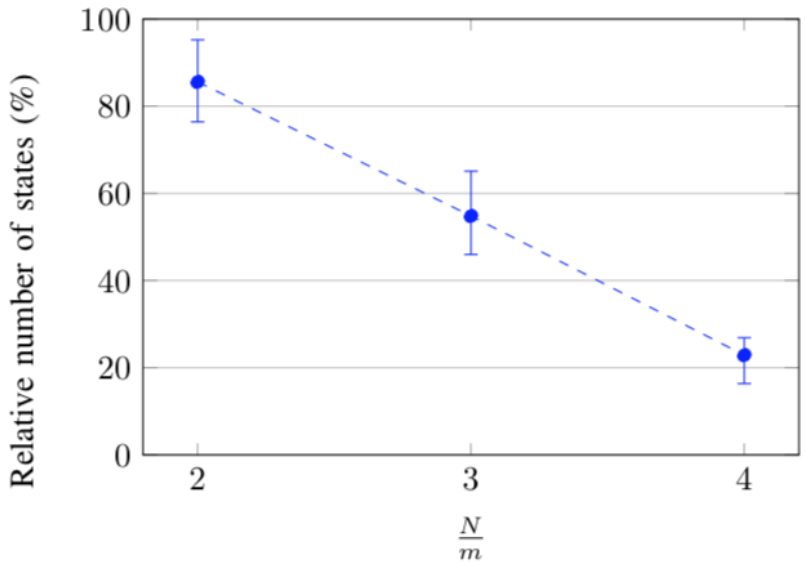

Figure – $N = 1$ to 12 tasks, $m = 1$ to 4 processors, $\beta^{max} = 20$.

From the first figure 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 $\frac{N}{m}$. The second figure 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.

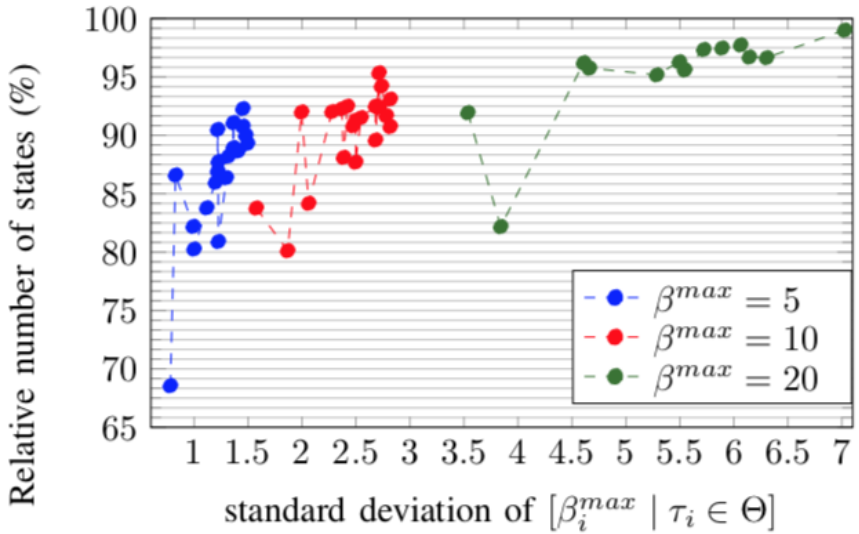

Figure – $N = 8$ tasks, $m = 4$ processors, $\beta^{max} \in \{5, 10, 20\}$.

The above figure 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.
It 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}.

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}.

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

Figure – EXECUTION TIME (IN SECONDS), WHEN $N = 16$ AND $m = 4$

The above table groups results for $N = 16$ and $m = 4$. In this case, the average execution time increases from 6.25 s to 385 s just by increasing β^{max} from 3 to 5.

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

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

The above table 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.

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

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

Lastly, the above table 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 previous tables, the influence of the individual β_i^{max} values on the overall execution time can also be seen : for example, with $N = 16$, $m = 4$ and $\beta^{max} = 5$, the execution time varies from 94 s to 599 s.

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

Figure – RESIDENT SET SIZE USED (IN MB), WHEN $\beta^{max} = 20$

Similar results are observed for memory occupation. Indeed, the whole state space has to be stored.

Observations :

- As expected, the time and space complexity of our algorithm makes it impossible to deal with systems that have too large a state space.

Observations :

- As expected, the time and space complexity of our algorithm makes it impossible to deal with systems that have too large a state space.
- Many industrial systems use small multicore platforms. For instance, the 32 bit Micro-controller TriCore family developed by Infineon for the embedded automotive market offers platforms with 1 to 6 cores.

Observations :

- As expected, the time and space complexity of our algorithm makes it impossible to deal with systems that have too large a state space.
- Many industrial systems use small multicore platforms. For instance, the 32 bit Micro-controller TriCore family developed by Infineon for the embedded automotive market offers platforms with 1 to 6 cores.
- 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.

Observations :

- As expected, the time and space complexity of our algorithm makes it impossible to deal with systems that have too large a state space.
- Many industrial systems use small multicore platforms. For instance, the 32 bit Micro-controller TriCore family developed by Infineon for the embedded automotive market offers platforms with 1 to 6 cores.
- 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.

Table of contents

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

In our work :

- 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.

In our work :

- 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.

In our work :

- 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 our experiments we observe two points :

1. 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.
2. the time complexity of our algorithm is too high to deal with anything but small systems.

- From the previous observations, 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.

- From the previous observations, 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.

- From the previous observations, 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.

Table of contents

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.

Thank your for your attention, questions ?